BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Natural Language Processing
(AIMLC ZC)

Dr YVK Ravi Kumar
yvk.ravikumar@pilani.bits-pilani.ac.in

These slides are prepared by the instructor, with grateful acknowledgement of James Allen and many others who made their course materials freely available online.

Session 2 & 3
(2nd and 9th August 2026)

Vector Semantics and Embedding

Agenda for the session

Vector Semantics and Word Embedding (*Ref: Chapter 6, Jurafsky and Martin***)**

- Lexical semantics
- Vector semantics
- Word and Vectors
- TF-IDF
- Word2Vec
  - Skip gram
  - CBOW
- GloVe
- Visualizing Embeddings

Lexical semantics

- **Structure** of words : **morphology**
- **Distribution** of words: **language modeling**
- **Meaning** of words: **lexical semantics**
- **Distributional hypothesis:**
  - Words that occur in **similar contexts** tend to have **similar meanings**
    E.g. oculist and eye-doctor occur near words like eye or examined
  - Amount of meaning difference between two words "corresponding roughly to the amount of difference in their environments (Context)"
  - two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.
  - **Basic idea: Measure the semantic similarity of words in terms of the similarity of the contexts in which they appear**

Applications

Question answering:

Q: "How ***tall*** is Mt. Everest?"

Ans: "The official ***height*** of Mount Everest is 29029 feet"

- ***"tall"* is similar to *"height"***

Plagiarism detection

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant

MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.

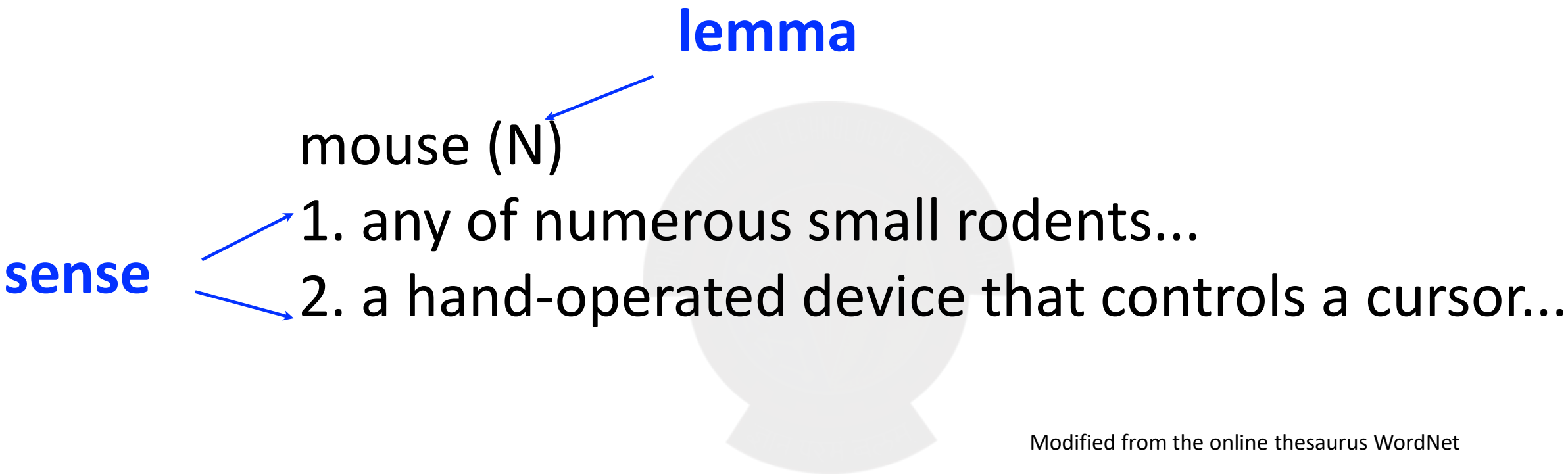
What do words mean?

- **N-gram** or text classification methods:
  - Words are just strings (or indices w_i in a vocabulary list)
  - That's not very satisfactory!
- **Introductory logic** classes:
  - The meaning of "dog" is DOG; cat is CAT

    $\forall x\ \mathrm{DOG}(x) \longrightarrow \mathrm{MAMMAL}(x)$

  - That seems hardly better!

Lemmas and senses



Modified from the online thesaurus WordNet

✓ A sense or "concept" is the meaning component of a word Lemmas can be polysemous (have multiple senses)

Lexical semantics

- **Lexical Semantics** is the linguistic study of **word meaning**.

- **Word Senses**
  - A word may have one or more **senses**
  - Word senses have a **many-to-many relationship** with words:
    - A word can have multiple senses (e.g., *bank*).
    - A concept can be expressed by multiple words (e.g., *doctor, physician*).

- **Semantic Relationships Between Words**
  - Synonymy, Similarity, Relatedness, Antonymy, Connotation

- A model of word meaning should enable an NLP system to **reason about meaning** and support tasks such as **question answering, information retrieval, and machine translation**.

Relations between senses: Synonymy

- **Synonyms** have the same meaning in some or all contexts.
  - Same word sense, substitutable for one another, same propositional meaning, truth preserving
    - couch/sofa, vomit/throw up, car/automobile
  - However:
    - water/H20 : "H20" in a surfing guide?
    - big/large: my big sister != my large sister
- In practice, the word synonym is therefore used to describe a relationship of approximate or rough synonymy

Relation: Word Similarity

- Words that are not synonyms, but share some element of meaning
  - (cat, dog) (car, bicycle) (cow, horse)
- The notion of Word Similarity is very useful in larger semantic tasks
  - Helps in computing how similar the meaning of two phrases or sentences are
  - Applications : question answering, paraphrasing, and summarization.
- One classic way to quantify word similarity has been to use humans to judge how similar one word is to another:

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

Relation: Word Relatedness

- Also called ‘Word Association’

- Words can be related in any way, perhaps via a semantic frame or field
  - coffee, tea: **similar**
  - coffee, cup: **related**, not similar

- Semantic field is a set of words that cover a particular semantic domain and bear structured relations with each other, e.g.
  - **Hospitals -** *surgeon*, *scalpel*, *nurse*, *anesthetic*, *hospital*
  - **Restaurants** - *waiter*, *menu*, *plate*, *food*, *menu, chef*
  - **Houses -** *door*, *roof*, *kitchen*, *family*, *bed*

Relation: Antonymy

- Antonyms are words with senses that are opposite with respect to only one feature of meaning

- Otherwise, they are very similar!
  - dark/light - short/long - fast/slow - rise/fall
  - hot/cold - up/down - in/out

- More formally: antonyms can
  - define a binary opposition or be at opposite ends of a scale
    - long/short, fast/slow
  - Be *reversive*:
    - rise/fall, up/down

Connotation

- Words have affective meanings
- How the meaning of a word relates to a writer's/reader's emotions, sentiments or evaluations
  - Positive emotion (**happy**) v/s Negative emotion (**sad**)
  - Positive evaluation (**great, love**) v/s Negative evaluation (**terrible, hate**)

Connotation

- Words typically vary along 3 affective dimensions:
  - **Valence**: the pleasantness of the stimulus
  - **Arousal**: the intensity of emotion provoked by the stimulus
  - **Dominance**: the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	**love**	1.000		**toxic**	0.008
	happy	1.000		**nightmare**	0.005
Arousal	**elated**	0.960		**mellow**	0.069
	frenzy	0.965		**napping**	0.046
Dominance	**powerful**	0.991		**weak**	0.045
	leadership	0.983		**empty**	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

Example (V-A-D)

Word	Valence	Arousal	Dominance
Joy	8.8	6.0	7.0
Excitement	8.1	8.5	7.2
Afraid	2.1	8.4	2.0
Depressed	1.7	2.0	1.1

Computational models of word meaning

Q: Can we build a robust computational model that represent word meaning?

A: Vector Semantics!

- Emerged from the convergence of two key ideas:
  1. We can define the meaning of words by their linguistic distribution – i.e. their environments (the words around them)
     [Zellig Harris (1954): If A and B have almost identical environments, we say that they are synonyms.]

  2. Connotation of a word can be represented as a point in multidimensional space

- The standard model to represent meaning in language processing

- Handles many NLP goals rather efficiently

What does recent English borrowing *ongchoi* mean?

- Suppose you see these sentences:
  - Ongchoi is delicious **sautéed with garlic**.
  - Ongchoi is superb **over rice**
  - Ongchoi **leaves** with salty sauces
- And you've also seen these:
  - …spinach **sautéed with garlic over rice**
  - Chard stems and **leaves** are **delicious**
  - Collard greens and other **salty** leafy greens
- Conclusion:
  - Ongchoi is a leafy green like spinach, chard, or collard greens
    - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Word embeddings

- Vector semantics represents each word as a **point in a high-dimensional semantic space**.
- This semantic space is learned from the **distribution of neighbouring (context) words**.
- The vectors used to represent words are called **word embeddings**.
- Words with **similar meanings** sit **close to one another** in the semantic space.
- Word embeddings form the foundation of almost every modern NLP system

two-dimensional (t-SNE) projection of embeddings

Intuition: why vectors?

- Consider a Sentiment Analysis task:
  - Using **word identities**:
    - A feature is the **identity of a word**
    - Example feature: *"The sentence contains the word **'terrible'**"*
    - The model only recognizes the **exact same word**
    - If the test set contains **awful** or **horrible**, the feature is not activated
  - Using **word embeddings**:
    - A feature is the **word's vector representation**
    - Example feature: *The sentence contains a word with embedding **[35, 22, 17, ...]***
    - In the test set, we may encounter a similar embedding **[34, 21, 14, ...]**
    - Since similar words have similar vectors, the model can **generalize to unseen but semantically similar words**

Word embeddings: types

1. **Frequency based** Embeddings (Count based)
   - A common, baseline approach
   - **Sparse, high-dimension** vectors
   - Words represented using **counts (or weighted counts) of surrounding words**
   - Vector dimensions correspond to **vocabulary words (**or **documents** in a corpus)
   - **Examples**: Count Vector, TF-IDF Vector, Co-Occurrence Vector

2. **Prediction based** Embeddings
   - **Dense, low-dimensional vectors**
   - Word representations are **learnt by predicting the surrounding context**
   - Similar words learn similar vector representations
   - Examples: Word2Vec (Skip-Gram, CBOW), Glove

Representing documents as Vectors

A **document** is any unit of text (e.g., a book, article, email, web page, or report).
A document is made up of many **words (terms)**.

In a Word-Document Matrix:

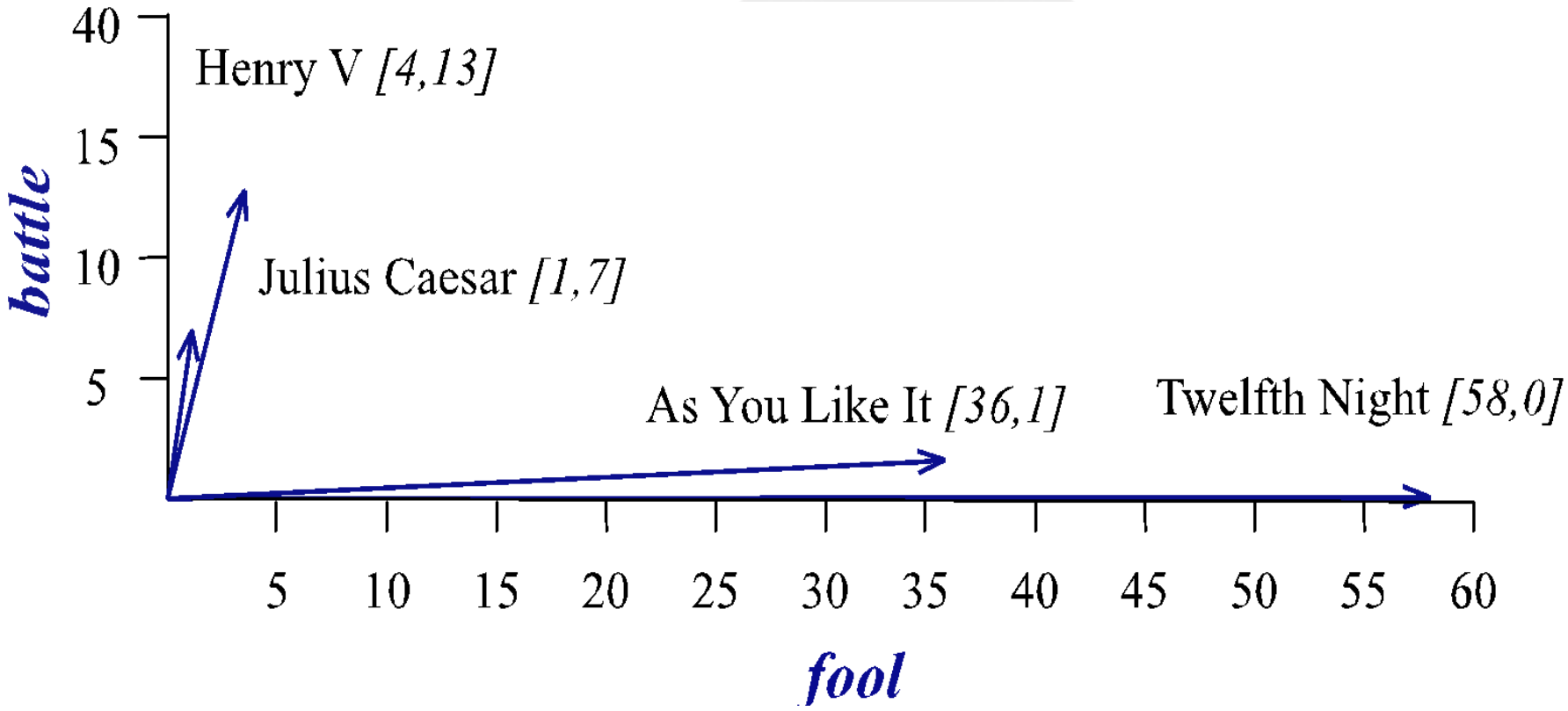
- Each **row** corresponds to a **word (term)**.
- Each **column** corresponds to a **document**.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Documents with **similar word distributions** have similar vectors.
  - For instance, the two **comedies** have similar vectors - they contain more occurrences of **fool** and **wit**.

Visualizing Document Vectors

Each document is represented as a point (or vector) in a multidimensional space.



From Document Vectors to Word Vectors

Instead of representing documents using words, we can also represent words using the documents in which they occur.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- ***battle*** is "*the kind of word that occurs in Julius Caesar and Henry V*"
- ***fool*** is "*the kind of word that occurs in comedies, especially Twelfth Night*"

Word-Word (Term-Context) matrix

Two words are considered semantically similar if their **context vectors** are similar.

In a **Word-Word matrix**, each cell captures the number of times the row (target) word and the column (context) word co-occur within some context window on some training corpus.

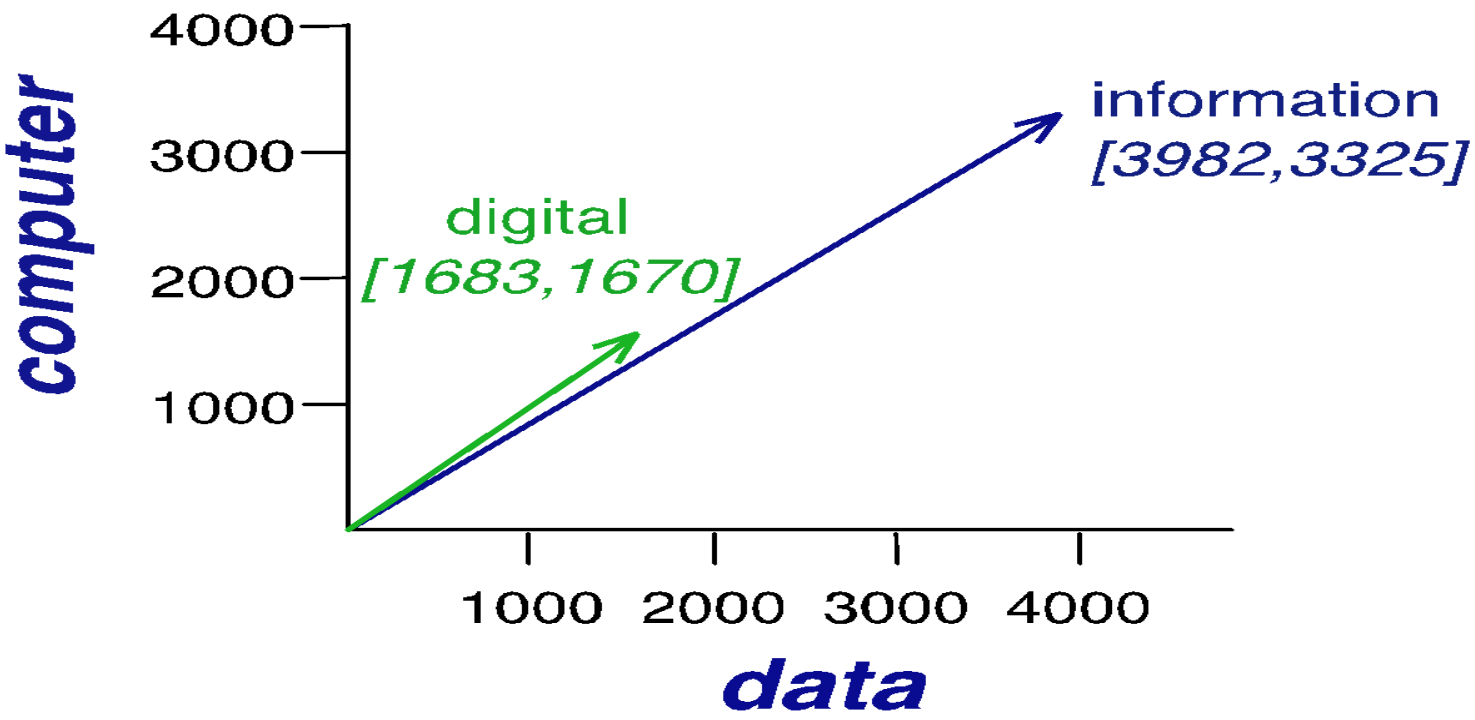
- Sample context windows (+/- 4 words):

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

- Sample Word-Word matrix:

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Visualizing Word Vectors



Computing word similarity: Dot product

The **dot product** between two vectors is a scalar:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- Dot product measures the similarity between two vectors.
- It is higher when two vectors have large values in the same dimensions.

context word

target word

	computer	data	fruit
digital	4	5	0
information	8	10	0
cherry	0	1	8

digital.information = 4.8 + 5.10 + 0.0
= **82**

digital.cherry = 4.0 + 10.1 + 0.8
= **10**

Problem with raw Dot Product

- Dot product is biased toward longer vectors.
- Longer vectors naturally produce larger dot products, even when the underlying meanings are not similar.
- Vector length: $|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$
- Frequent words (of, the, you) have long vectors, dot product undesirably favors frequent words.

context word

target word	computer	data	fruit
digital	4	5	0
information	8	10	0
cherry	0	1	8
the	40	25	30

digital.information = 4.8 + 5.10 + 0.0
= **82**

digital.cherry = 4.0 + 10.1 + 0.8
= **10**

digital.the = 4.40 + 5.25 + 0.30
= 285

Cosine Similarity: A better similarity measure

Cosine similarity normalizes the dot product by the lengths of the two vectors.

It measures the **angle** between vectors rather than their magnitude.

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta$$

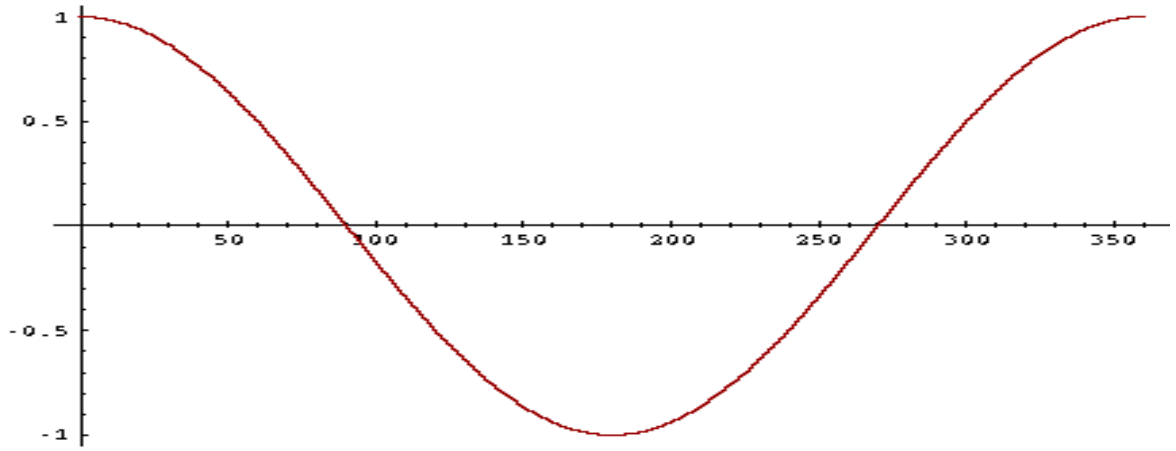
$$\frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine as a similarity metric

Cosine Similarity:

- **+1** => Vectors point in the same direction (maximum similarity)
- **0** => Vectors are orthogonal (no similarity)
- **–1** => Vectors point in opposite directions (opposite directions)



- Since frequency-based vectors contain only non-negative values, cosine similarity typically ranges from **0 to 1**.

Cosine example

context word

target word

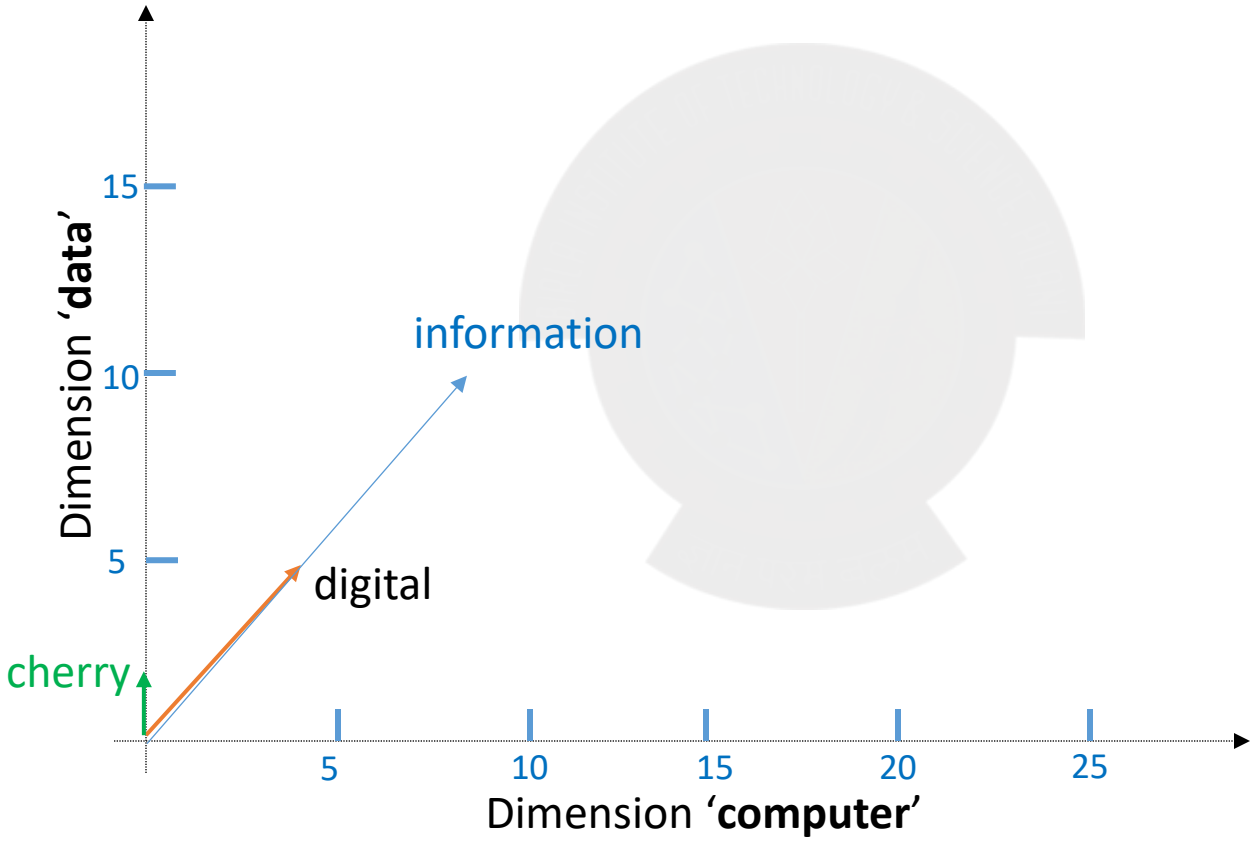
	computer	data	fruit
digital	4	5	0
information	8	10	0
cherry	0	1	8
the	40	25	30

$$\cos(\text{digital}, \text{information}) = \frac{(4 \times 8) + (5 \times 10) + (0 \times 0)}{\sqrt{4^2 + 5^2 + 0^2} \times \sqrt{8^2 + 10^2 + 0^2}} = \frac{82}{\sqrt{41} \times \sqrt{164}} = \boxed{1.000}$$

$$\cos(\text{digital}, \text{cherry}) = \frac{(4 \times 0) + (5 \times 1) + (0 \times 8)}{\sqrt{4^2 + 5^2 + 0^2} \times \sqrt{0^2 + 1^2 + 8^2}} = \frac{5}{\sqrt{41} \times \sqrt{65}} = \boxed{0.097}$$

$$\cos(\text{digital}, \text{the}) = \frac{(4 \times 40) + (5 \times 25) + (0 \times 30)}{\sqrt{4^2 + 5^2 + 0^2} \times \sqrt{40^2 + 25^2 + 30^2}} = \frac{285}{\sqrt{41} \times \sqrt{3125}} = \boxed{0.796}$$

Visualizing Cosine Similarity



tf-idf

- **Raw Frequency is a Poor Representation**
  - The co-occurrence matrices we have seen represent each cell using **raw word frequencies**.
  - Frequency is useful: if **sugar** frequently appears near **apricot**, it indicates a meaningful relationship.
  - However, very common words such as **the**, **it**, and **they** occur frequently in almost every context. Such high-frequency words contribute little to distinguishing the meaning of a document or a word.
- **How do we balance these two requirements?**
  - Reward words that are frequent within a document.
  - Down-weight words that are common across the entire corpus.
- **When the vector dimensions represent documents, we use tf-idf weighting**

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Term frequency (tf)

- **Term frequency** captures the frequency of the word ***t*** in the document ***d***.
- One could use raw counts here:

$$\mathbf{tf}_{t,d} = \text{count}(t,d)$$

- However, a $\log_{10}$ of the raw count is used more often:

$$\mathbf{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$$

Document frequency (df)

- **Document Frequency (df_t)** of a term is the number of documents in which the term ***t*** appears.
- df_t is **different from Collection Frequency (cf_t)**, which counts the total number of occurrences of the term across the entire collection.

Example: Shakespeare's 37 Plays

- Consider the words Romeo and action:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

- ***Romeo*** is a much better word for identifying documents about Romeo and Juliet.
- ***action*** appears in many plays and therefore carries less discriminative power.

Inverse document frequency (idf)

- **Inverse Document Frequency (idf_t)** of a term measures how informative or discriminative the term ***t*** is.
  - Words that appear in **fewer documents** receive a **higher idf**.
  - Words that appear in **many documents** receive a **lower idf**.

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

where:

- **N** = Total number of documents in the corpus
- **df_t** = Document Frequency of term *t*

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Combining TF and IDF: The tf-idf Weight

tf-idf: $w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$

A word receives a **high tf-idf score** if it:

- appears **frequently in a document** (high tf), *and*
- appears in **few documents overall** (high idf).

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Word *wit* for document *As You Like it*

tf = $\log_{10}(20+1)$ = 1.322, idf = $\log_{10}(37/34)$ = 0.037, tf-idf value = **0.049**

Word *good* for document *As You Like it*

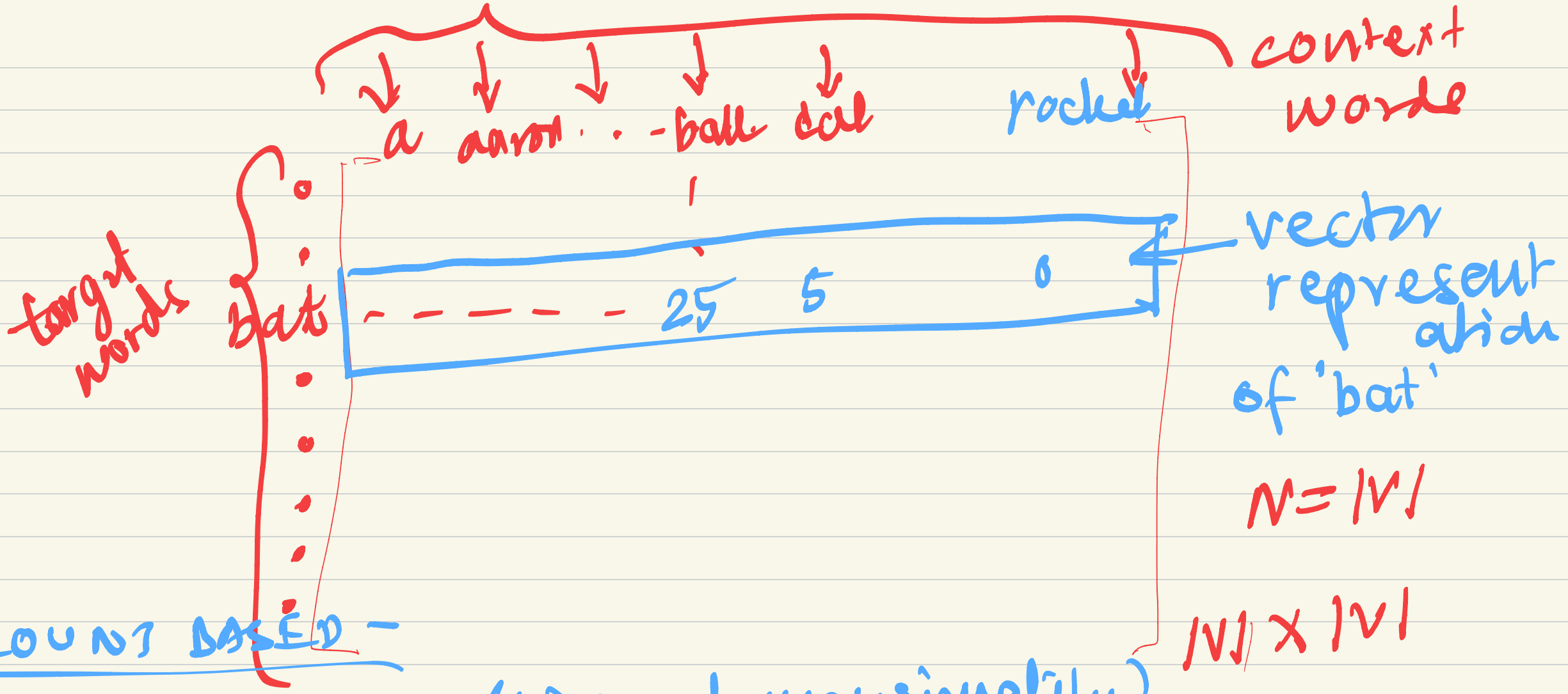
tf = $\log_{10}(114+1)$ = 1.322, idf = $\log_{10}(37/37)$ = 0.0, tf-idf value = **0.0**

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Sparse v/s Dense Vectors

- **Sparse Vectors**, produced by count/frequency based methods**:**
  - **High-dimensional** (one dimension per vocabulary word or document, |V| = 20,000 to 50,000)
  - Most entries are **0** (sparse)
  - Similar words (e.g., *car* and *automobile*) are treated as independent features

- Can there be an alternative - **Dense Vectors?**
  - **Low-dimensional** (typically **50–1000 dimensions**)
  - Most entries contain **non-zero real-valued numbers**
  - Similar words have **similar vector representations**


a
aaron
ball
zoo
rocket
context words
target words
bat
25
5
0
vector representation of 'bat'
N = |V|
|V| x |V|
COUNT BASED -
• too long (high-dimensionality)
• sparse
|V| ≈ 50,000

Methods for learning Dense Vector Embeddings

Static Word Embeddings

- *A single vector is learned for each word, regardless of context.*
- Word2Vec
  - Skip-Gram
  - CBOW
- GloVe

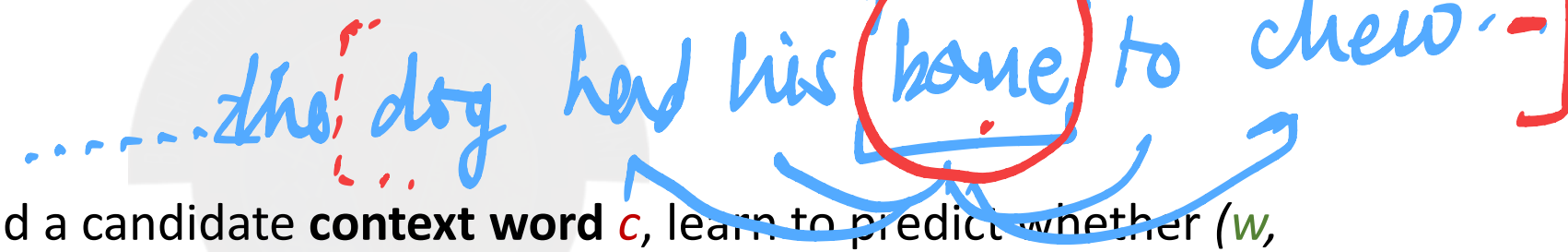
Contextual Word Embeddings

- *A word's embedding depends on the sentence in which it appears, leading to **different embeddings for the same word** in different contexts.*
- ELMo
- BERT

Learning Word Embeddings by Predicting Context

From counting to predicting

- **Frequency-based methods -** Represent a word using counts of its neighbouring words
- **Word2Vec -** Instead of explicitly counting co-occurrences, learn word vectors by predicting whether words occur in each other's context



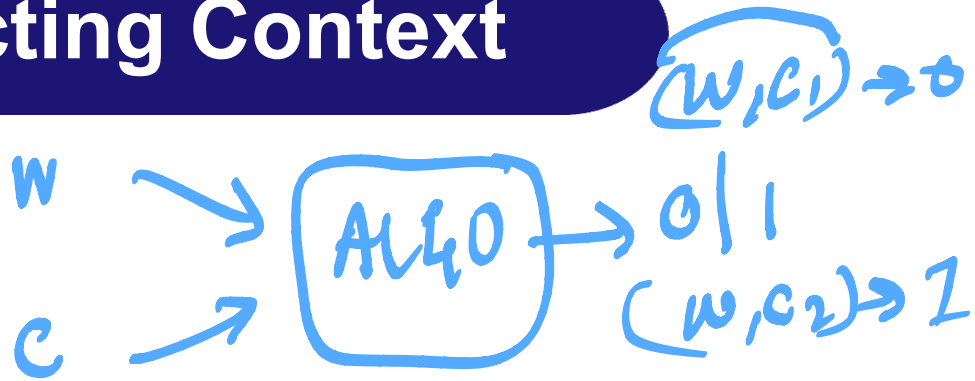
Key idea

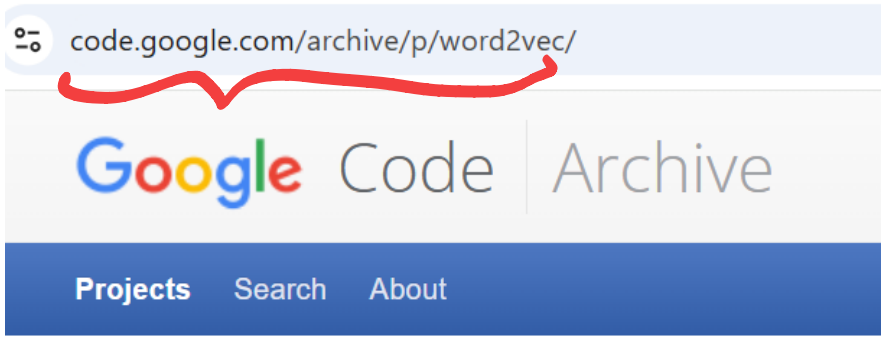
- Given a **centre word** *w* and a candidate **context word** *c*, learn to predict whether *(w, c)* is a **genuine context pair**
- Words that occur in similar contexts are pushed toward **similar vector representations**
- The prediction task is only a means to an end, the key output is the **learned word embeddings**

Learning Word Embeddings by Predicting Context

Another cool idea – Self-supervision

- Learns without human labels, the text corpus itself provides the training signal
  - Observed *(centre, context)* pairs : **positive examples**
  - Randomly sampled pairs : **negative examples**





Two main Word2Vec architectures:

- Skip-Gram: given centre word w, predict its context words c
- CBOW: given context words c, predict the center word w

Project

Source

word2vec

Training Methods:

- Negative Sampling
- SoftMax

① skipgram + NS → SGNS

② cBow + softmax

Words as vectors – Representations

One-hot representation

- Represent each word using a vector with one dimension for every word in the vocabulary
- |V| dimensions, king:queen not much different than king:apple

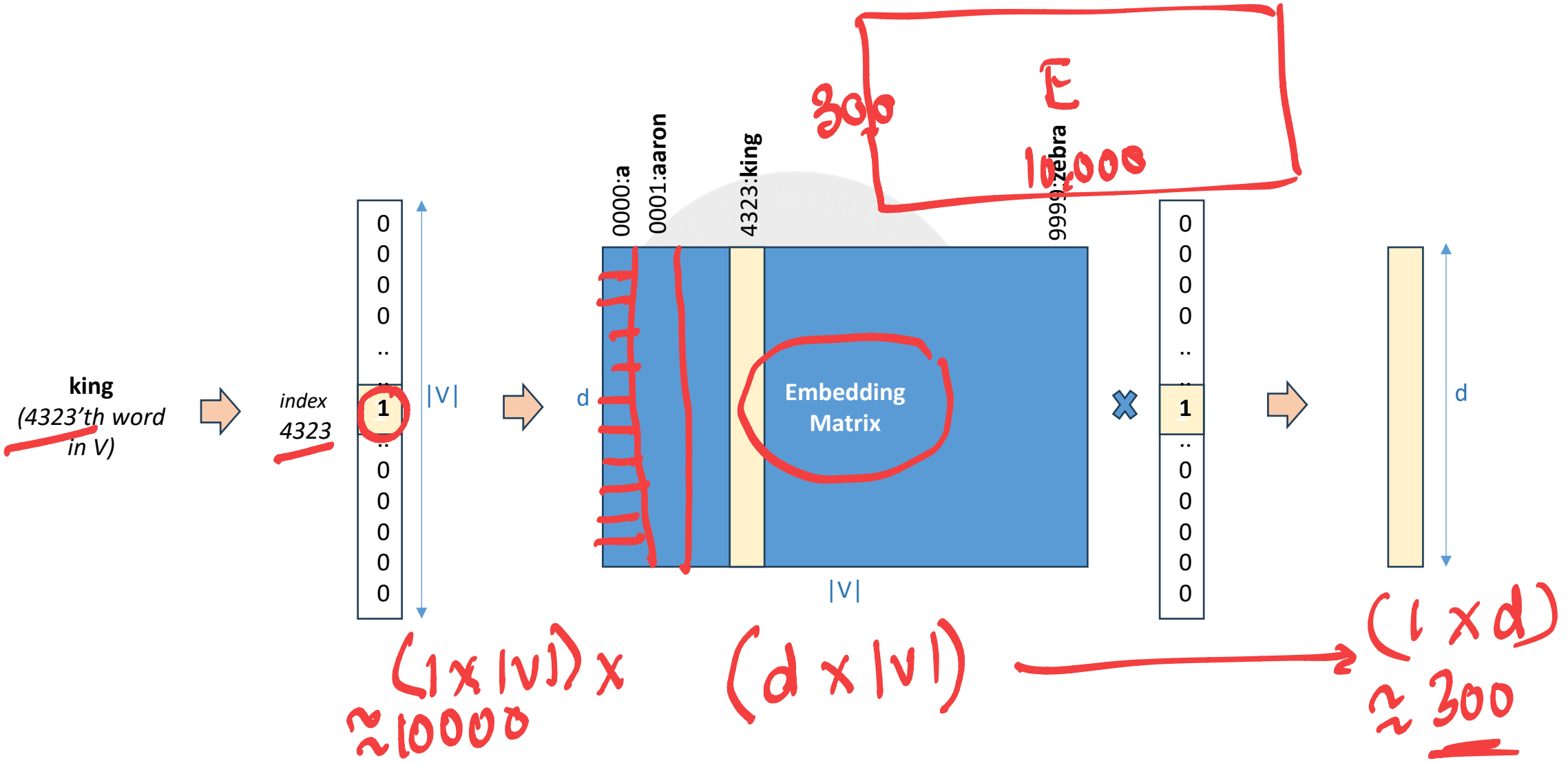
man	1	0	0	0	0	0
woman	0	1	0	0	0	0
king	0	0	1	0	0	0
queen	0	0	0	1	0	0
apple	0	0	0	0	1	0
orange	0	0	0	0	0	1

Featurized representation

- represent a word using a small set of meaningful features
- Semantically meaningful, but who defines features for |V| words?

	Gender	*Royal*	*Age*	*Food*
man	-1	0.01	0.03	0.09
woman	1	0.02	0.02	0.01
king	-0.95	0.93	0.70	0.02
queen	0.97	0.95	0.69	0.01
apple	0.00	-0.01	0.03	0.95
orange	0.01	0.00	-0.02	0.97

Embedding Matrix



word2vec–SGNS: Strategy

1. Process the n-word window around the target word ***t*** in running text
2. Treat ***t*** and a neighboring context word ***c*** as **positive examples**
3. Randomly sample other words in the lexicon to get **negative examples**
4. Use logistic regression to train a classifier to distinguish those two cases
5. Use the learned weights of the classifier as the embeddings

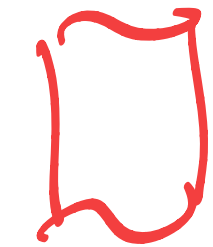
Let's talk through…

1. Some preliminaries
   - Binary classification
   - Logistic Regression
   - Gradient Descent

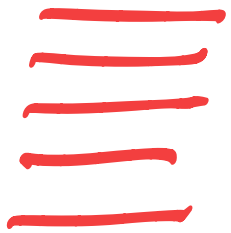
2. word2vec SGNS Intuition

3. word2vec SGNS formulation

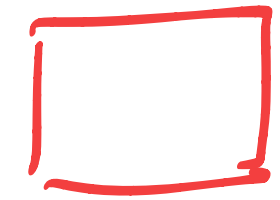
(1) Binary Classification



→ spam/non-spam



→ loan worthy/not loan worthy



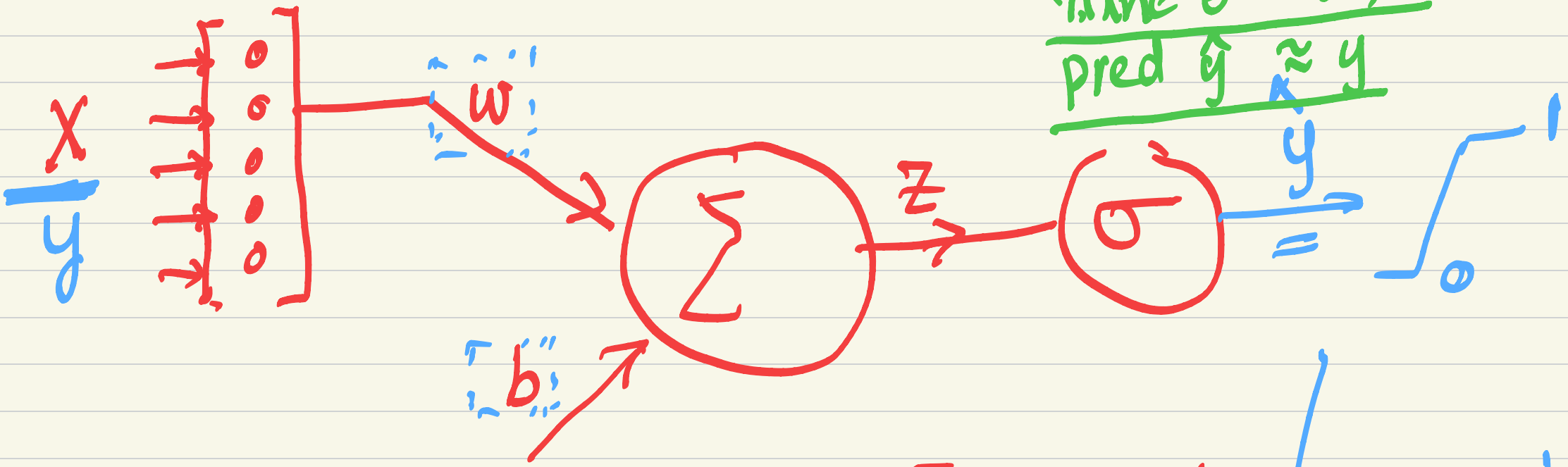
→ cat/not-cat

(2) Logistic Regression

Given x, y

Find $\theta = w, b$

Pred $\hat{y} \approx y$

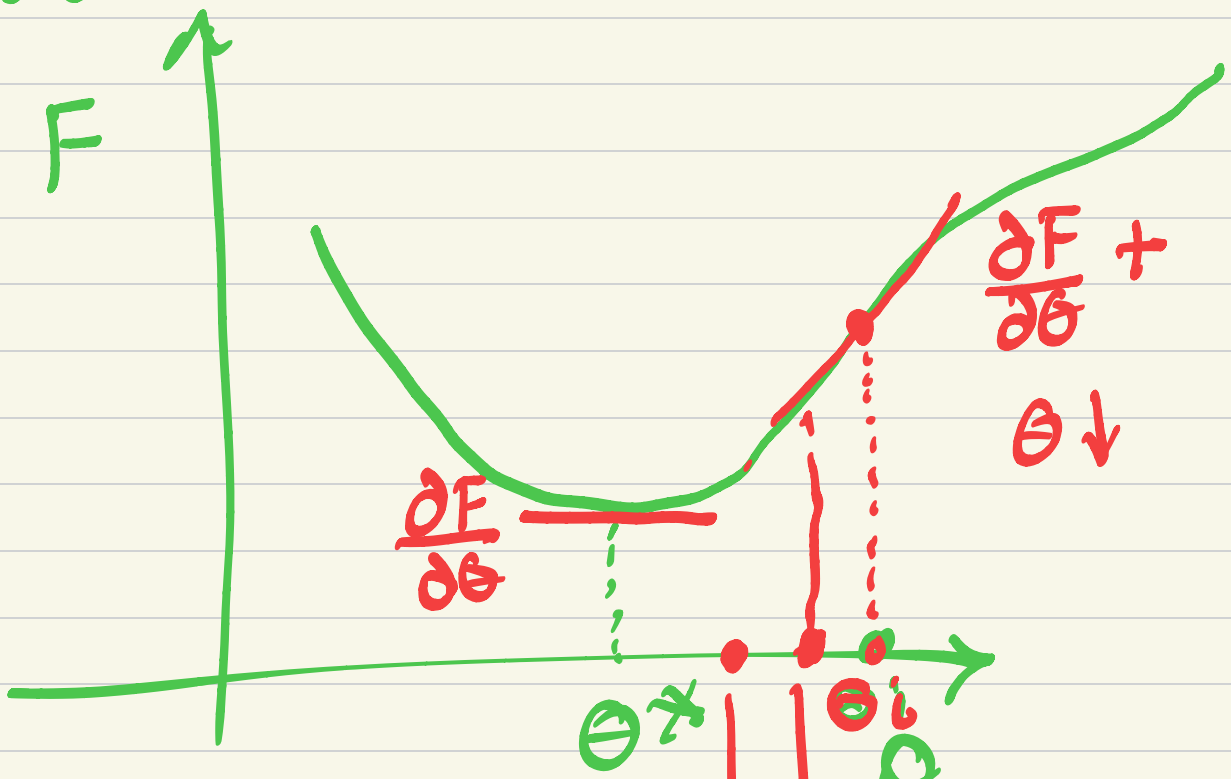


$$z = [w_1\ w_2\ w_3 \ldots w_N] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + b$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\int_{-\infty}^{\infty} \rightarrow \int_0^1$$

(3) Gradient Descent



$F(\theta)$

goal: find θ^* such that

$F(\theta^*) < F(\theta')$

L = Loss fn $= L(\theta)$

$\theta^0 \rightarrow \theta^1 \rightarrow \theta^2 \rightarrow \theta^n$

$F(\theta^n)$

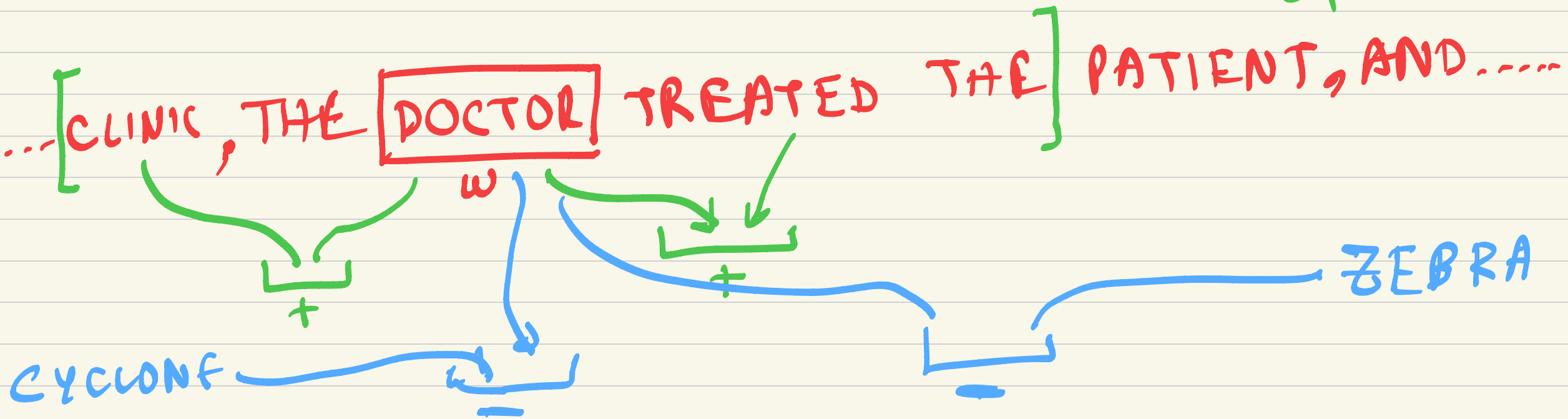
$$\theta_{new} = \theta_{old} - \eta \nabla_\theta F$$

Word2vec

w → doctor

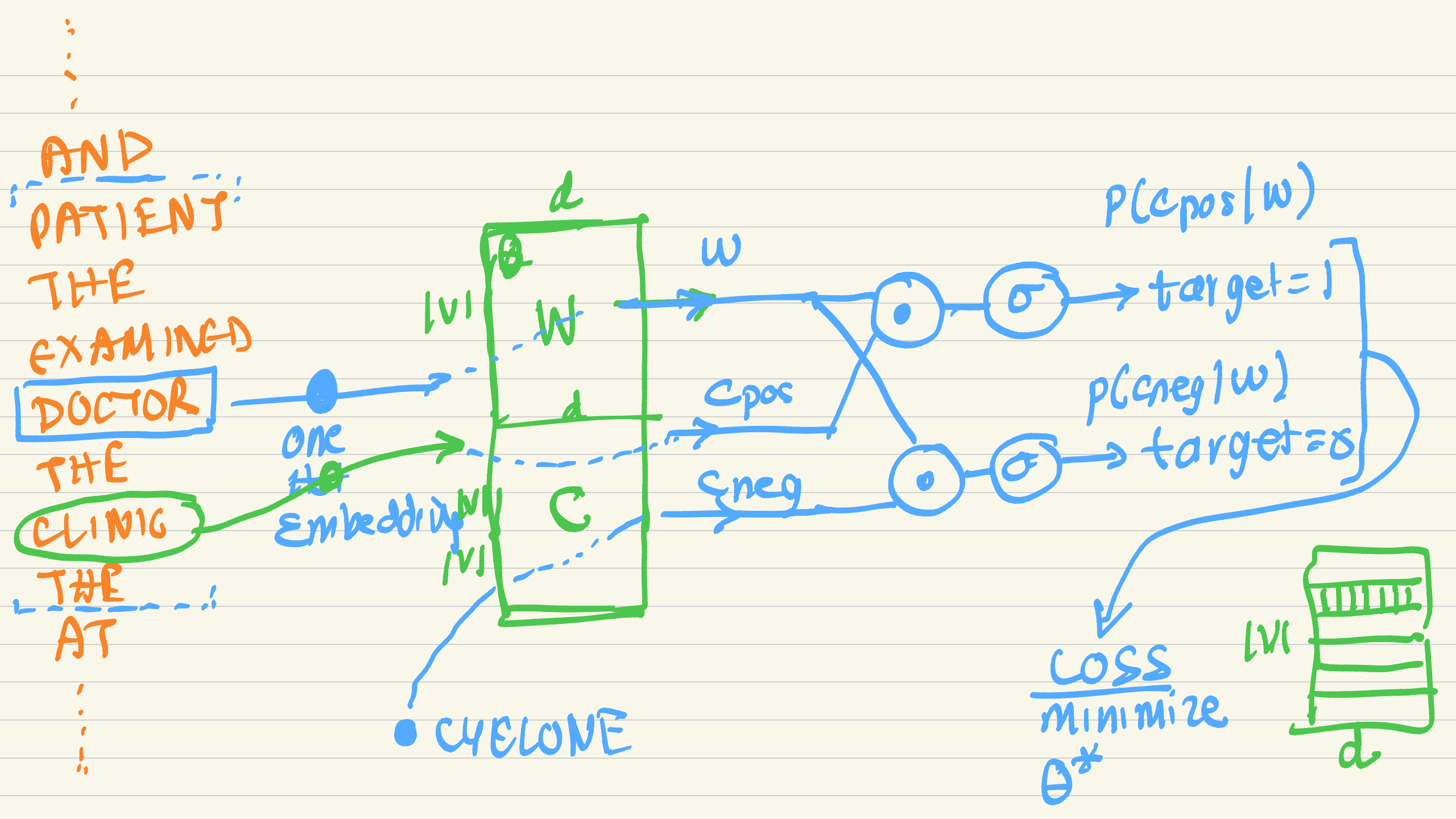
w'

train a classifier: "is w' a context word of w"



WORD2VEC
SGNS

1. PICKED A TARGET WORD and CREATED A WINDOW AROUND IT
2. + pairs (TARGET WORD, a word inside window)
3. − pairs (-do- , a random word from vocab)
4. Train a logistic classifier (w, c) → 0/1
5. Use parameters of the classifier as embeddings.


AND
PATIENT
THE
EXAMINED
DOCTOR
THE
CLING
THE
AT
one
Embeddi
d
|V|
W
d
C
|V|
w
cpos
cneg
CYCLONE
p(cpos|w)
target=1
p(cneg|w)
target=0
Loss
minimize
θ*
|V|
d

word2vec SGNS

① every + example → k negative examples

② WEIGHTED UNIGRAM FREQUENCY

w_1 .05 w_2 .0005

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} (count(w'))^\alpha}$$

③ $w, c \rightarrow w, w+c, \left(\frac{w+c}{2}\right)$

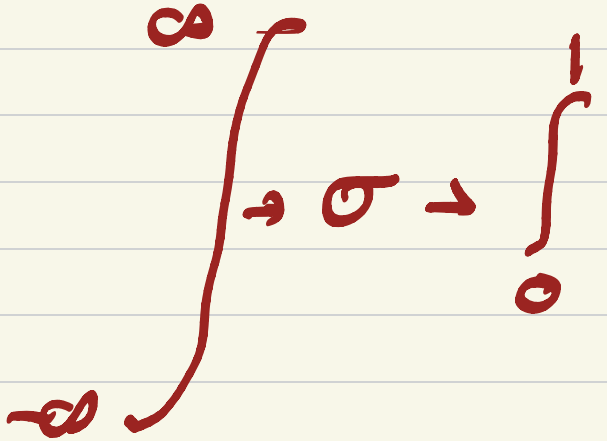
MATHEMATICAL FORMULATION

Input : tuple (w, c) → context word
└→ center word

Output : $P(+|w, c)$ → c is a + context word for w

$P(-|w, c) \rightarrow 1 - P(+|w, c)$

How do we compute $p(+ \mid w, c)$?



- similarity$(w, c) = c \cdot w$
- $p(+ \mid w, c) = \sigma(c \cdot w)$

  $p(- \mid w, c) = 1 - p(+ \mid w, c)$

In one training iteration: w, c_{pos}, k c_{neg}

$$L = -\log \left[p(+ \mid w, c_{pos}) \cdot \prod_{k} p(- \mid w, c_{neg}) \right]$$

GRADIENTS

$$\frac{\partial L}{\partial c_{pos}}, \frac{\partial L}{\partial c_{neg}}, \frac{\partial L}{\partial w}$$

SGD step:

$$c_{pos}^{(t+1)} \leftarrow c_{pos}^{(t)} + f\left(\frac{\partial L}{\partial c_{pos}}\right)$$

$$c_{neg}^{(t+1)} \leftarrow c_{neg}^{(t)} + \dots$$

$$w^{(t+1)} \leftarrow w^{(t)} + \dots$$

Word2vec: SG-NS

Word2vec

- An unsupervised NLP method developed by Google in 2013 (Mikolov et al. 2013)
- Quantify the relationship between words
- Framework for learning word vectors Idea:
  - We have a large corpus (“body”) of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context (“outside”) words o
  - Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:

[target]

…lemon, a [tablespoon of apricot jam, a] pinch…

- c1 c2 c3 c4

Skip-Gram Classifier

- (assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

…

And assigns each pair a probability:

$P(+|w, c)$

$P(-|w, c) = 1 - P(+|w, c)$

Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity(w,c) $\propto w \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

Turning dot products into probabilities

- Sim(w,c) ≈ w · *c*
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier computes *P*(+|*w*, *c*)

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

- This is for one context word, but we have lots of context words.
- We'll assume independence and just multiply them:

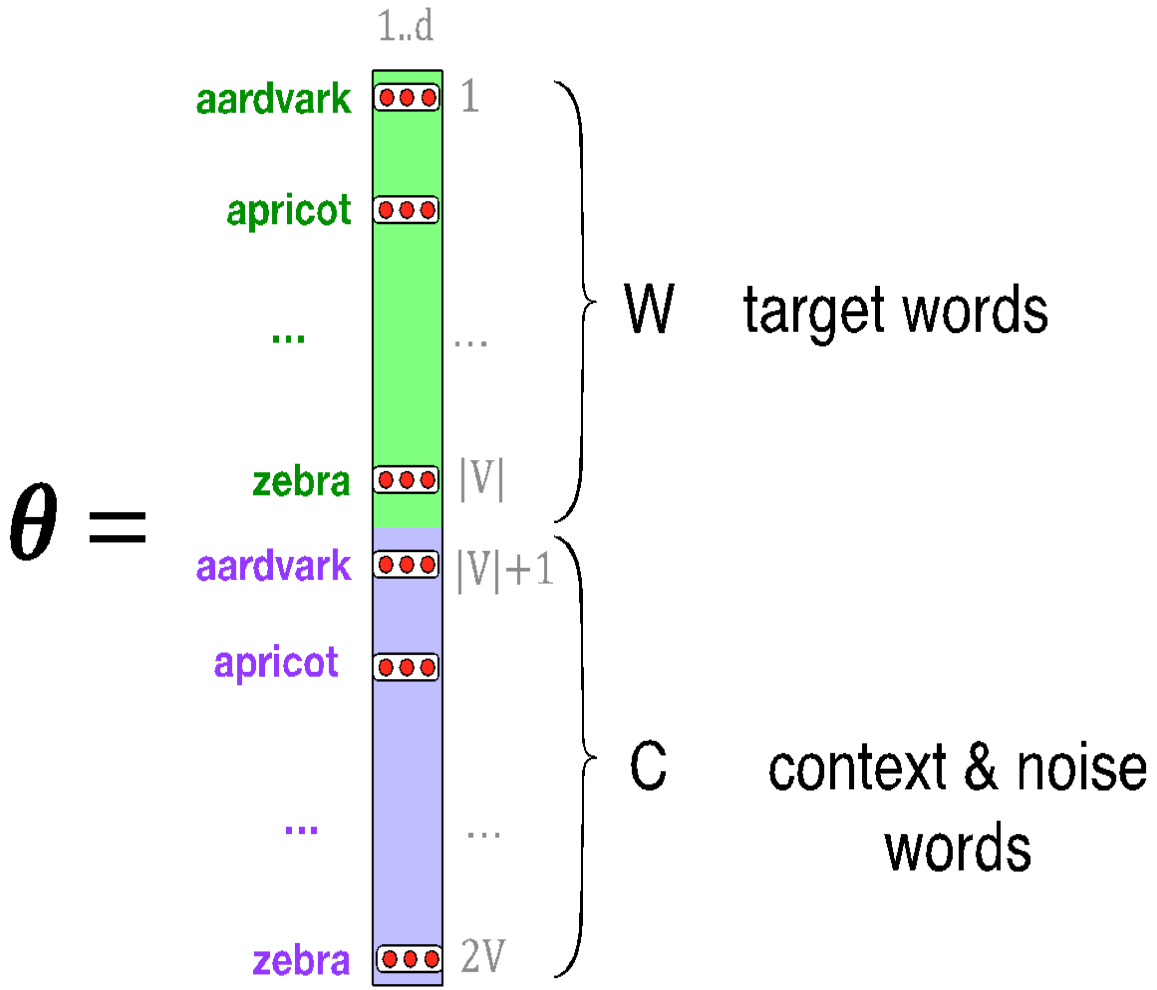
$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

- A probabilistic classifier, given
  - a test target word *w*
  - its context window of *L* words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).
- To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c

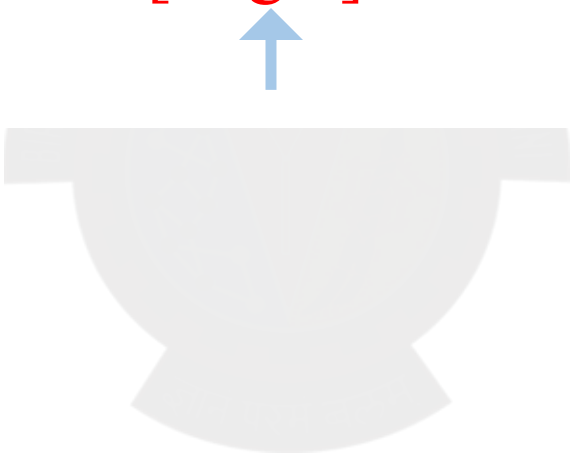


Skip-Gram Training data

- For every + example select k negative examples

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4



positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

The paper (Mikolov et al., 2013) says that K=2 ~ 5 works for large data sets, and K=5 ~ 20 for small data sets.

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

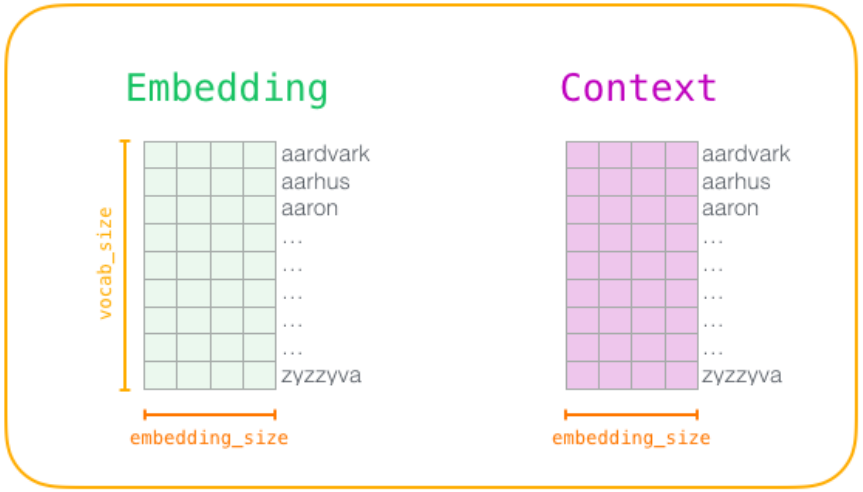
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

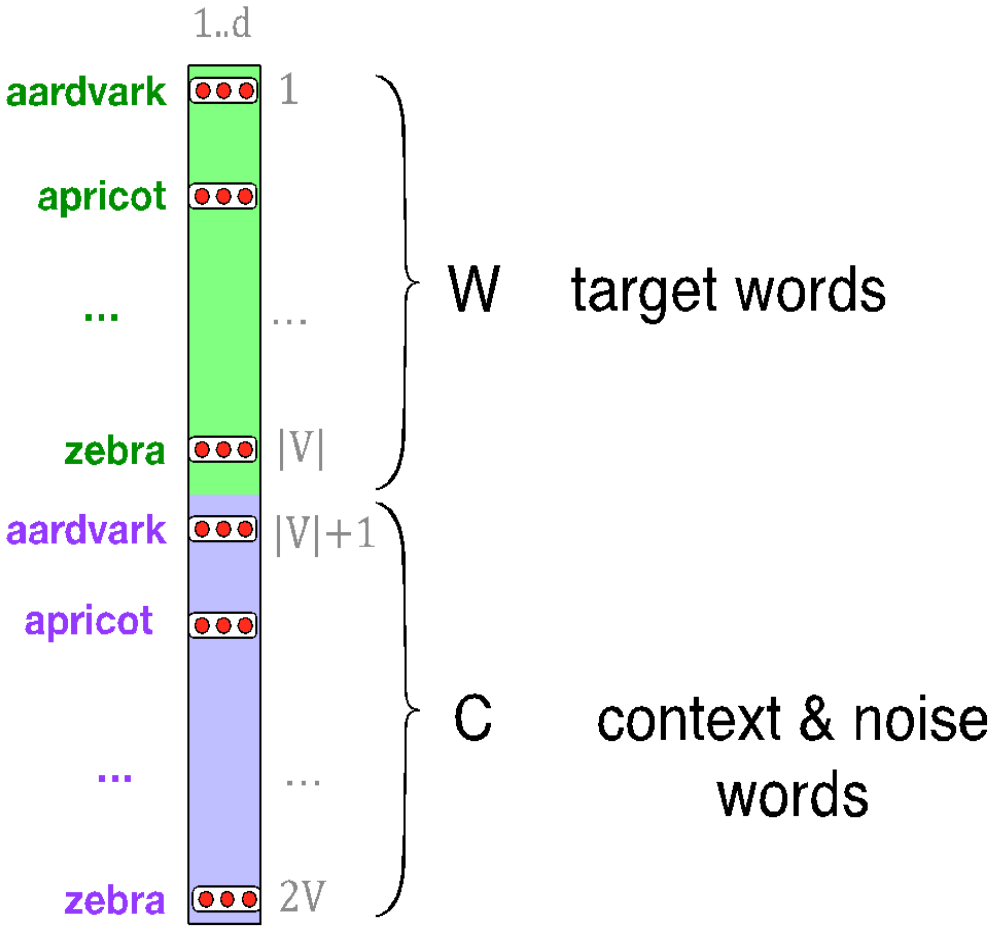
t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Skip-gram parameters

- SGNS learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C
- It's common to just add them together, representing word i as the vector $w_i + c_i$
- Thus the parameters we need to learn are two matrices W and C, each containing an embedding for every one of the |V| words in the vocabulary |V|



$\theta =$



Word2vec training :how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs (w , c_{pos}) drawn from the positive data
  - **Minimize** the similarity of the (w , c_{neg}) pairs drawn from the negative data.

•Log Loss function

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Loss function for one *w with* c_{pos} , c_{neg1}

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

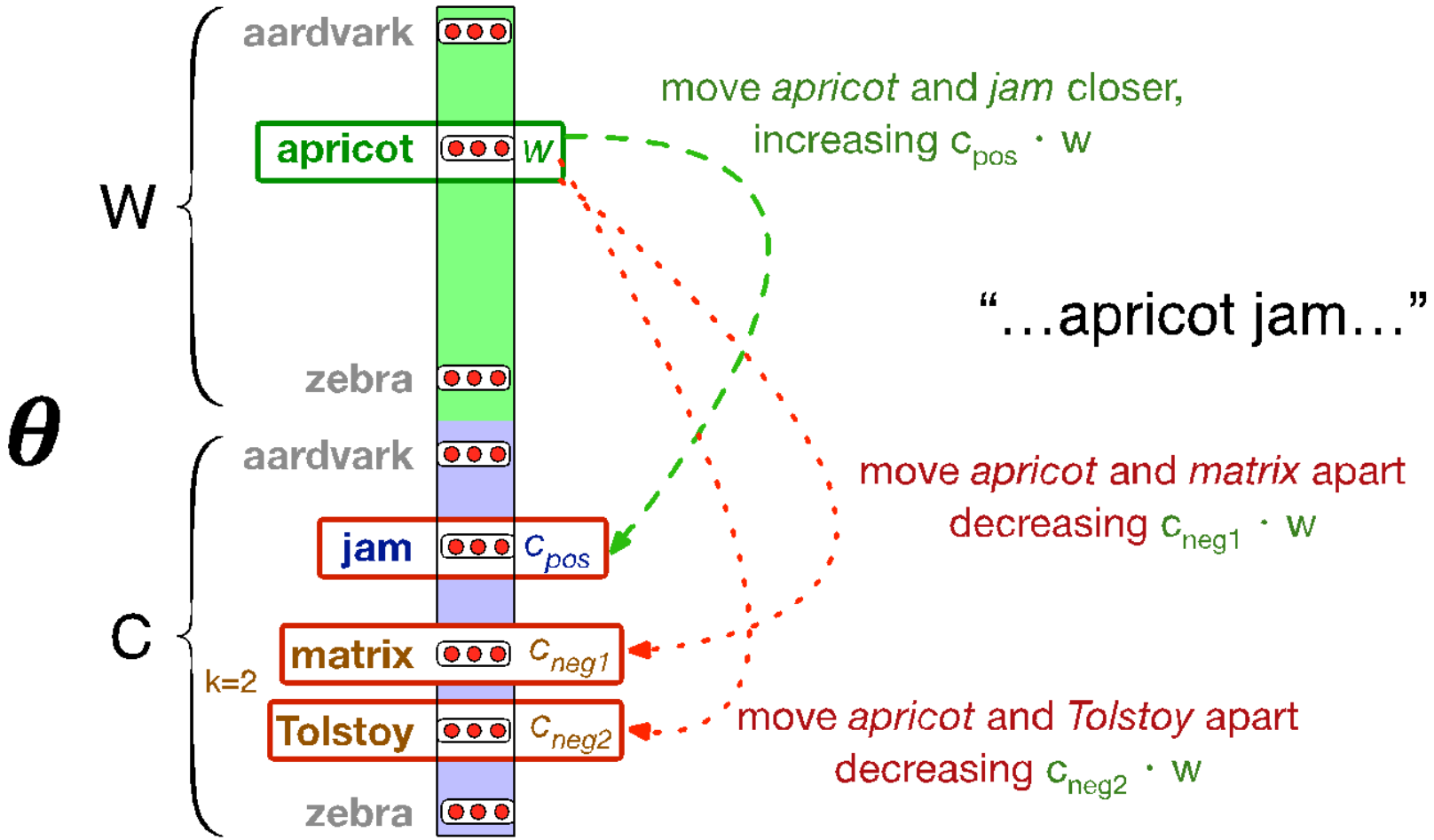
$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

Intuition of one step of gradient descent



- Tries to shift embeddings
- So the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam)
- And further from (lower dot product with) context embeddings for noise words that don't occur nearby (here Tolstoy and matrix).

Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
  - Higher learning rate means move *w* faster

$$w^{t+1} = w^{t} - h \frac{d}{dw} L(f(x; w), y)$$

The derivatives of the loss function

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^{k} [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$

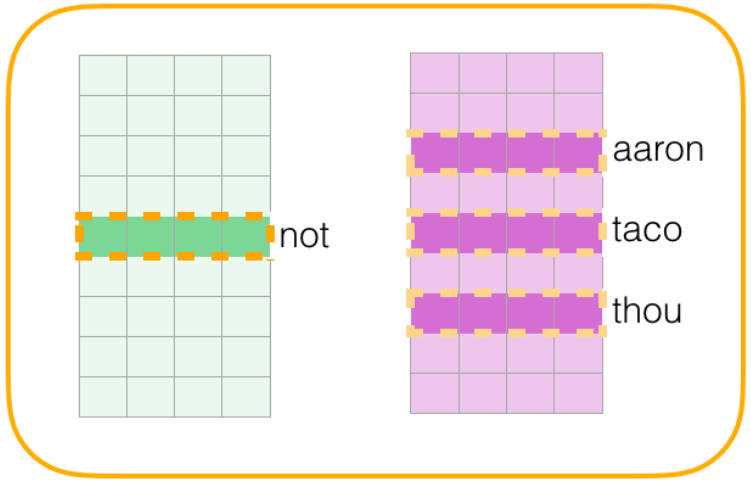
$$w^{t+1} = w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i}\right]$$

•Skip-gram training example

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$
$$= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}$$



Update Model Parameters

https://jalammar.github.io/illustrated-word2vec/

Two sets of embeddings

- SGNS learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C
- It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

Word Embeddings

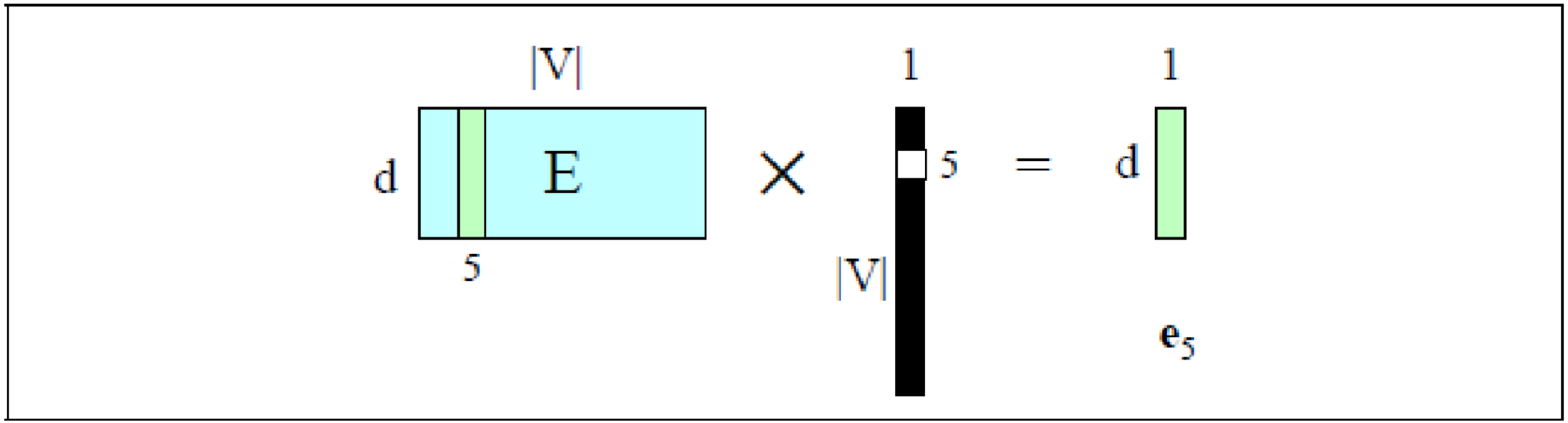


Figure 7.12 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

Example

- Corpus = "Ned Stark is the most honourable man"
- Current word-context pair = ("Ned", "Stark")
- Current negative words = "pimples", "zebra", "idiot"

	one hot vector				
ned	1	0	0	0	0
stark	0	1	0	0	0
pimples	0	0	1	0	0
zebra	0	0	0	1	0
idiot	0	0	0	0	1

initial embedding matrix			
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

initial context matrix			
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

Initial embeddings

Current word-context pair = ("*Ned*", "*Stark*")

Current negative words = "*pimples*", "*zebra*", "*idiot*"

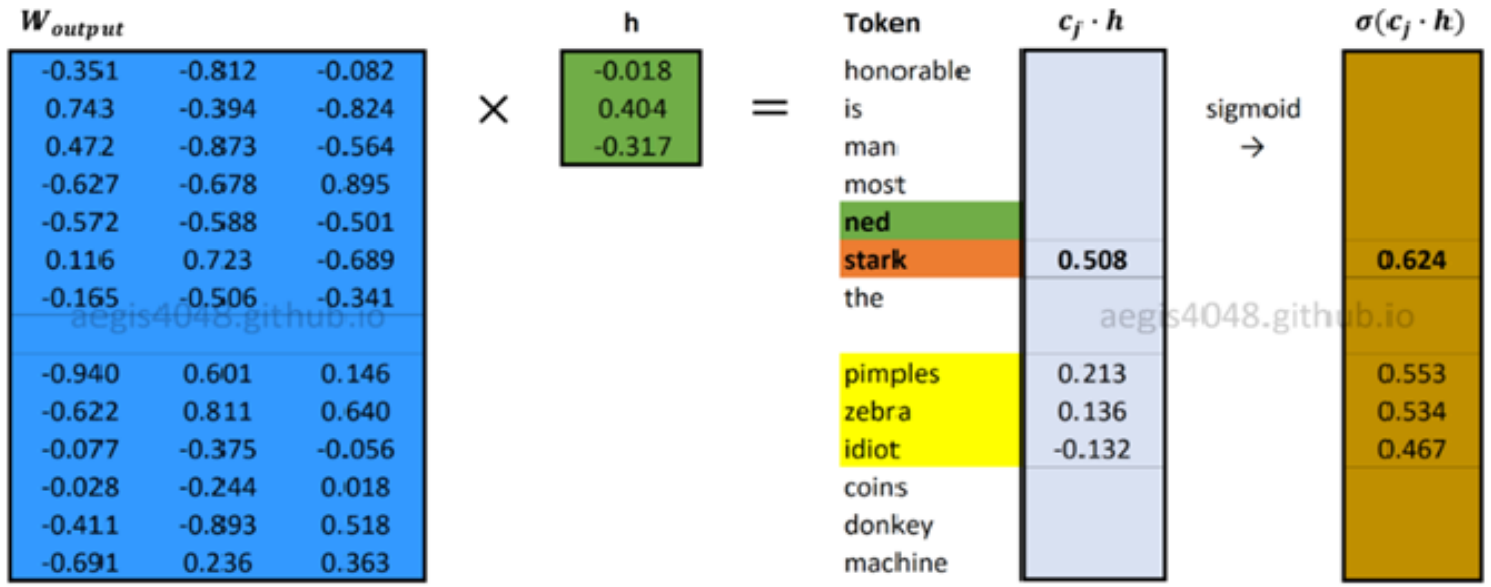
#	Token	x	W_{input}			h
0	honorable	0	-0.014	0.135	0.109	-0.018
1	is	0	-0.665	0.669	0.309	0.404
2	man	0	0.702	0.621	-0.981	-0.317
3	most	0	0.214	-0.813	-0.561	
4	**ned**	**1**	-0.018	0.404	-0.317	
5	**stark**	**0**	0.204	-0.007	-0.733	
6	the	0	-0.652	-0.097	0.499	
Neg.	pimples	0	-0.706	0.745	0.002	
	zebra	0	-0.492	-0.709	0.133	
	idiot	0	-0.628	-0.073	0.502	
	coins	0	0.456	0.767	-0.629	
	donkey	0	0.348	0.544	-0.740	
	machine	0	-0.627	0.487	0.466	

aegis4048.github.io

	input embeddings initial		
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

Forward Propagation: Sigmoid output layer

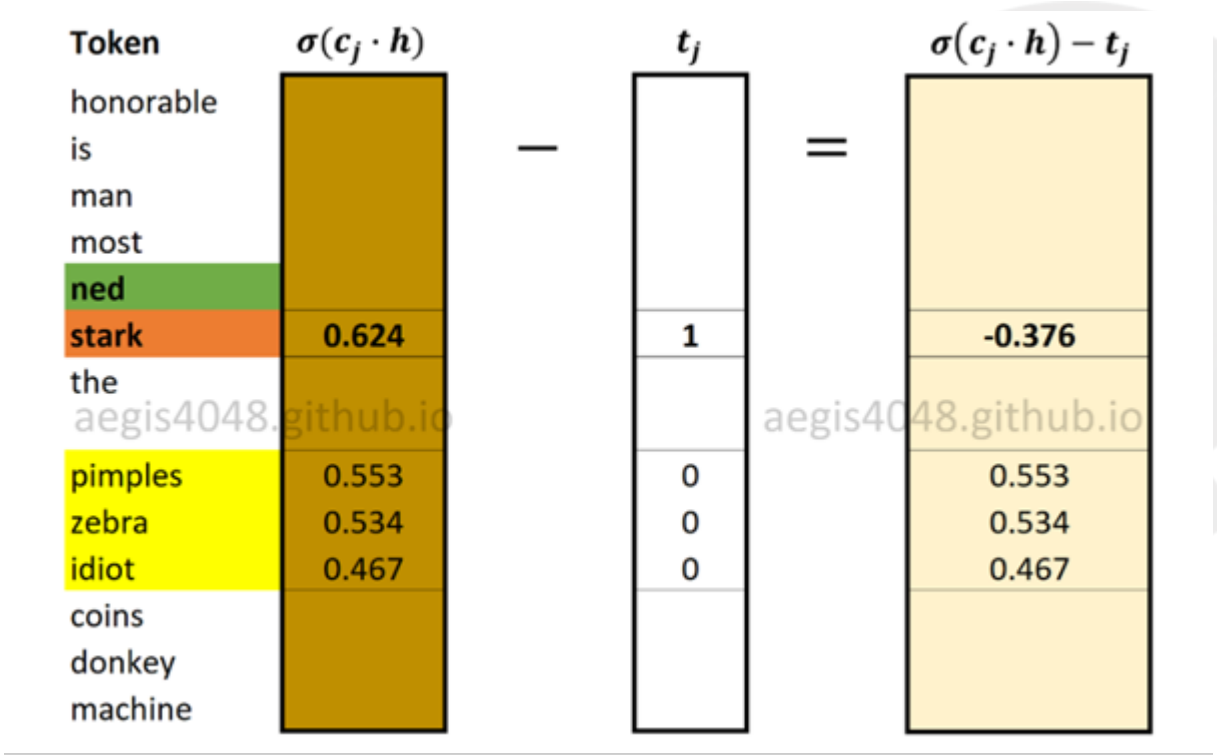


$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

old context embeddings				old i/p word embedding	dot product	sigmoid()
stark	0.116	0.723	-0.689	-0.018	0.508	0.624
pimples	-0.94	0.601	0.146	0.404	0.213	0.553
zebra	-0.622	0.811	0.64	-0.317	0.136	0.534
idiot	-0.077	-0.375	-0.056		-0.132	0.467

Backward Propagation: Prediction Error

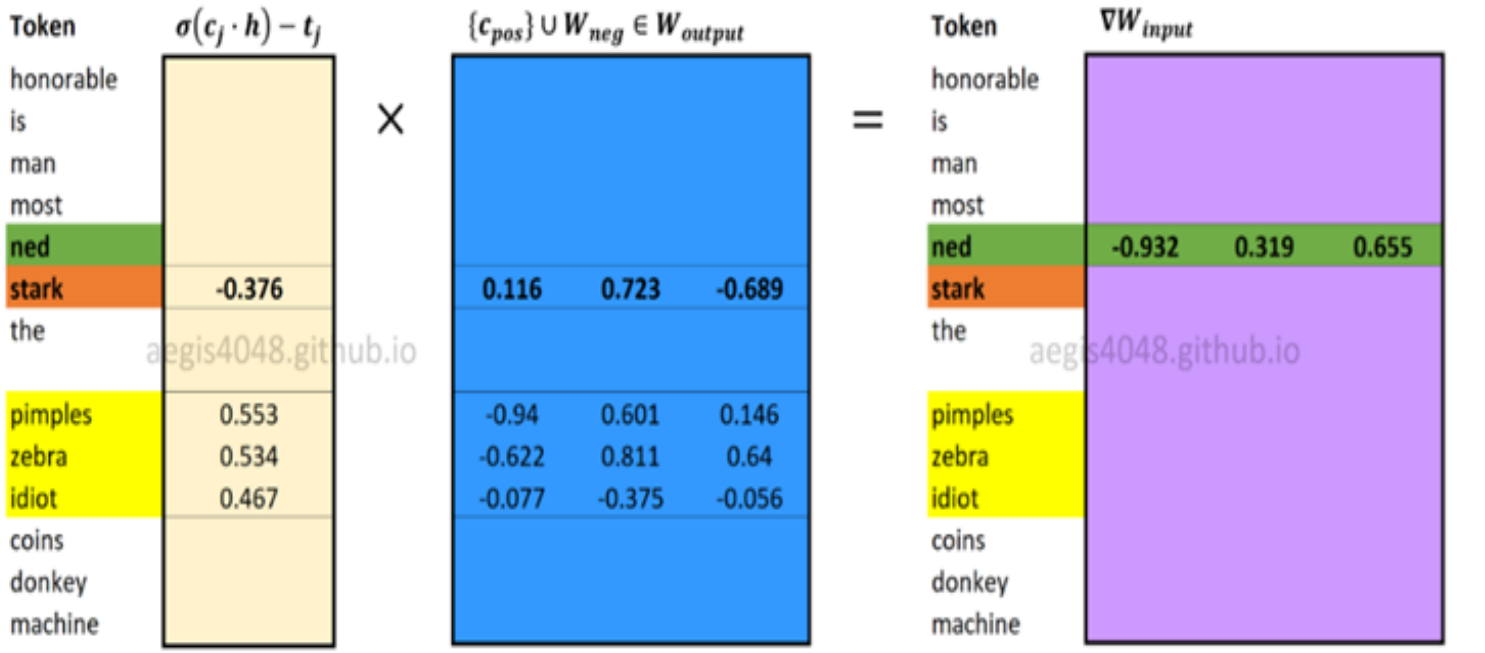
- our current positive word is Stark, t_j=1 for Stark and t_j=0 for other negative words (pimples, zebra, idiot).



aegis4048.github.io

Sigmoid()	t	sig-t (pred error)
0.624	1	-0.376
0.553	0	0.553
0.534	0	0.534
0.467	0	0.467

Derivative of loss w.r.t input word embeddings



$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k} [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i}$$

Sigmoid()	t	sig-t (pred error)
0.624	1	-0.376
0.553	0	0.553
0.534	0	0.534
0.467	0	0.467

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

sig-t(pred error)	C*(sig-t)			derivative of loss w.r.t input word embeddings		
-0.376	-0.04362	-0.271848	0.259064	-0.93154	0.318454	0.65541
0.553	-0.51982	0.332353	0.080738			
0.534	-0.33215	0.433074	0.34176			
0.467	-0.03596	-0.175125	-0.02615			

Input word embedding update

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k} [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i} \right]$$

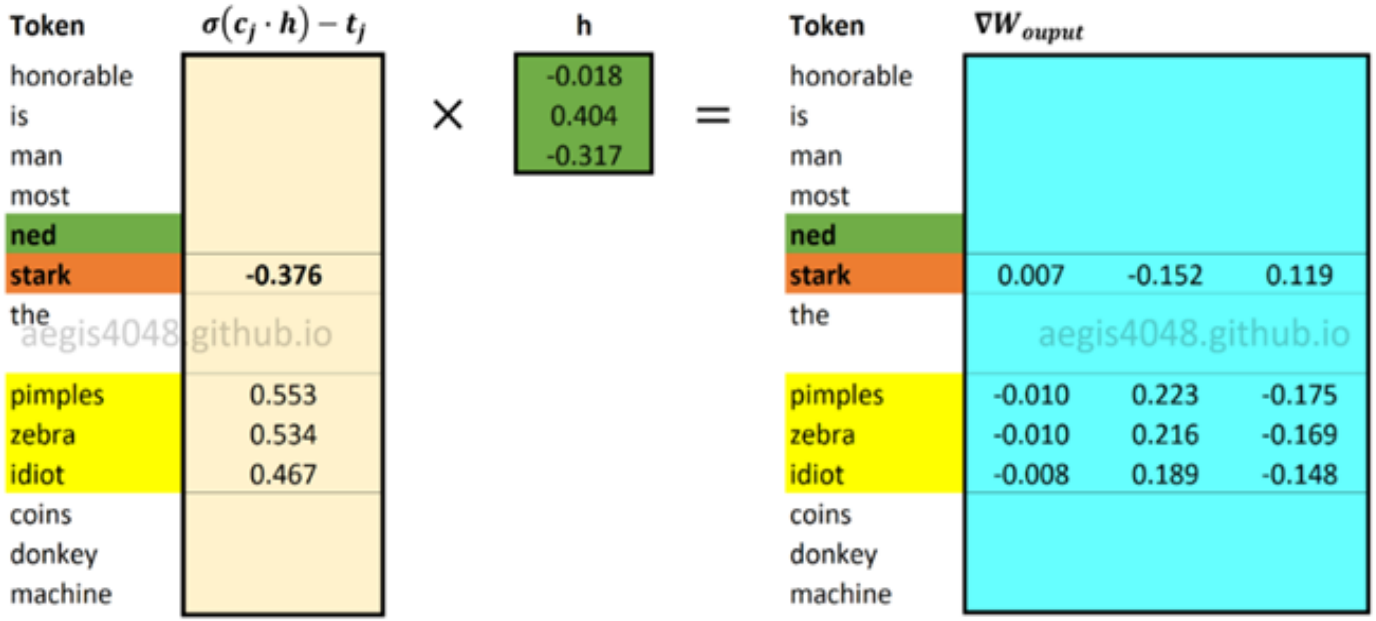
Token	$W_{input}^{(old)}$				η		∇W_{input}				$W_{input}^{(new)}$		
honorable	-0.014	0.135	0.109	−	0.05	×				=	-0.014	0.135	0.109
is	-0.665	0.669	0.309								-0.665	0.669	0.309
man	0.702	0.621	-0.981								0.702	0.621	-0.981
most	0.214	-0.813	-0.561								0.214	-0.813	-0.561
ned	-0.018	0.404	-0.317				**-0.932**	**0.319**	**0.655**		**0.029**	**0.388**	**-0.350**
stark	0.204	-0.007	-0.733								0.204	-0.007	-0.733
the	-0.652	-0.097	0.499								-0.652	-0.097	0.499
pimples	-0.706	0.745	0.002								-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133								-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502								-0.628	-0.073	0.502
coins	0.456	0.767	-0.629								0.456	0.767	-0.629
donkey	0.348	0.544	-0.740								0.348	0.544	-0.740
machine	-0.627	0.487	0.466								-0.627	0.487	0.466

aegis4048.github.io

Derivative of loss w.r.t input word embeddings		
-0.93154	0.318454	0.65541

word	old input word embedding			LR	LR* derivative w.r.t input			new i/p word embedding		
ned	-0.018	0.404	-0.317	0.05	-0.04658	0.015923	0.032771	0.029	0.388	-0.35

Derivative of loss w.r.t context word embeddings



$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos} \cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

Sigmoid-t (pred error)	i/p word embedding	Derivative of loss w.r.t context word embeddings		
-0.376	-0.018	0.007	-0.152	0.119
0.553	0.404	-0.01	0.223	-0.175
0.534	-0.317	-0.01	0.216	-0.169
0.467		-0.008	0.189	-0.148

Context word embedding update

Token	$W^{(old)}_{output}$			η		∇W_{ouput}			=	$W^{(new)}_{output}$		
honorable	-0.351	-0.812	-0.082	−	0.05	×			=	-0.351	-0.812	-0.082
is	0.743	-0.394	-0.824							0.743	-0.394	-0.824
man	0.472	-0.873	-0.564							0.472	-0.873	-0.564
most	-0.627	-0.678	0.895							-0.627	-0.678	0.895
ned	-0.572	-0.588	-0.501							-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689				0.007	-0.152	0.119	0.116	0.731	-0.695
the	-0.165	-0.506	-0.341							-0.165	-0.506	-0.341
pimples	-0.94	0.601	0.146				-0.010	0.223	-0.175	-0.940	0.590	0.155
zebra	-0.622	0.811	0.64				-0.010	0.216	-0.169	-0.622	0.800	0.648
idiot	-0.077	-0.375	-0.056				-0.008	0.189	-0.148	-0.077	-0.384	-0.049
coins	-0.028	-0.244	0.018							-0.028	-0.244	0.018
donkey	-0.411	-0.893	0.518							-0.411	-0.893	0.518
machine	-0.691	0.236	0.363							-0.691	0.236	0.363

aegis4048.github.io

$$c^{t+1}_{pos} = c^{t}_{pos} - \eta\,[\sigma(c^{t}_{pos} \cdot w^{t}) - 1]\,w^{t}$$
$$c^{t+1}_{neg} = c^{t}_{neg} - \eta\,[\sigma(c^{t}_{neg} \cdot w^{t})]\,w^{t}$$

word	old context word embedding			LR	LR* derivative w.r.t context			new context word embedd		
stark	0.116	0.723	-0.689	0.05	0.00035	-0.0076	0.00595	0.116	0.731	-0.695
pimples	-0.94	0.601	0.146	0.05	-0.0005	0.01115	-0.00875	-0.94	0.59	0.155
zebra	-0.622	0.811	0.64	0.05	-0.0005	0.0108	-0.00845	-0.622	0.8	0.648
idiot	-0.077	-0.375	-0.056	0.05	-0.0004	0.00945	-0.0074	-0.077	-0.384	-0.049

References

- https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling#pred_error

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin

Window Size determines the kind of neighbours

Small context window (C = +/-2)

- Words occurring very close to the target tend to capture **syntactic similarity**.
- **Example: *Hogwarts***
  - Nearest neighbours are other fictional schools: *Sunnydale, Evernight, Blandings*
  - They occur in similar local grammatical contexts.

Large context window (C = +/-5)

- Words occurring in a broader context tend to capture **semantic relatedness**.
- **Example: *Hogwarts***
  - Nearest neighbours are words related to the **Harry Potter world**: *Dumbledore, half-blood*
  - They occur in the same broader semantic context or semantic field.

Word2vec-CBOW

CBOW (Continuous Bag of Words)

- Use the surrounding context words to predict the center word.

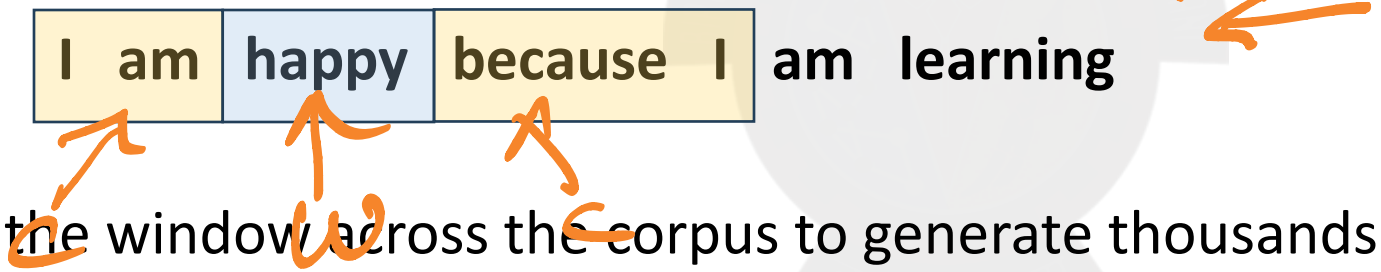
- Example: I am [happy] because I am learning
  - Context: **I, am, because, I**
  - Target: **happy**

CBOW: From Corpus to Training Data

Corpus:

I am happy because I am learning.

Using a window of size 5 (Context half-size, C = 2):

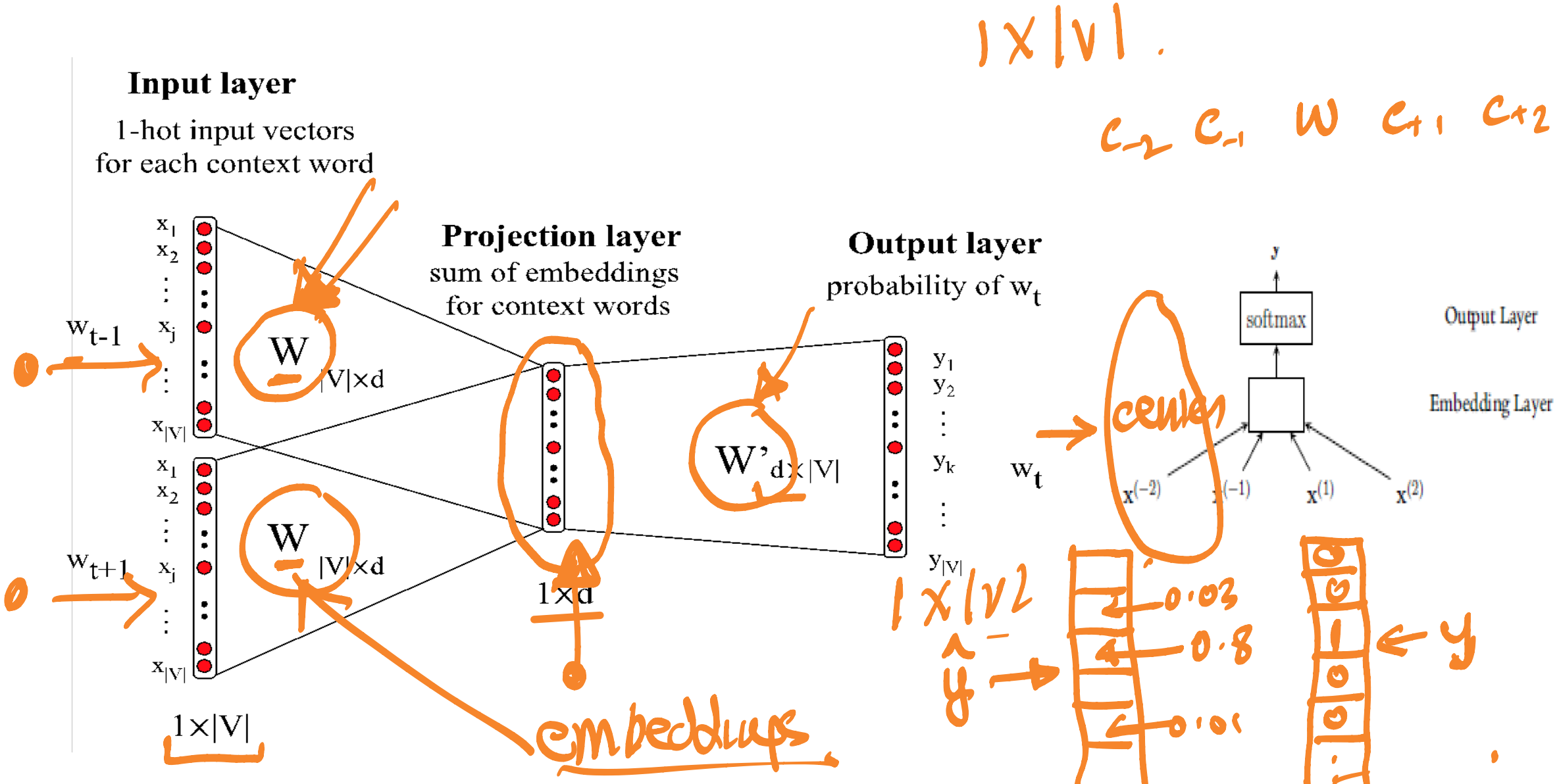


Slide the window across the corpus to generate thousands of training examples.

I am happy because I am learning

I am happy because I am learning

CBOW: Architecture



Skip gram versus CBOW

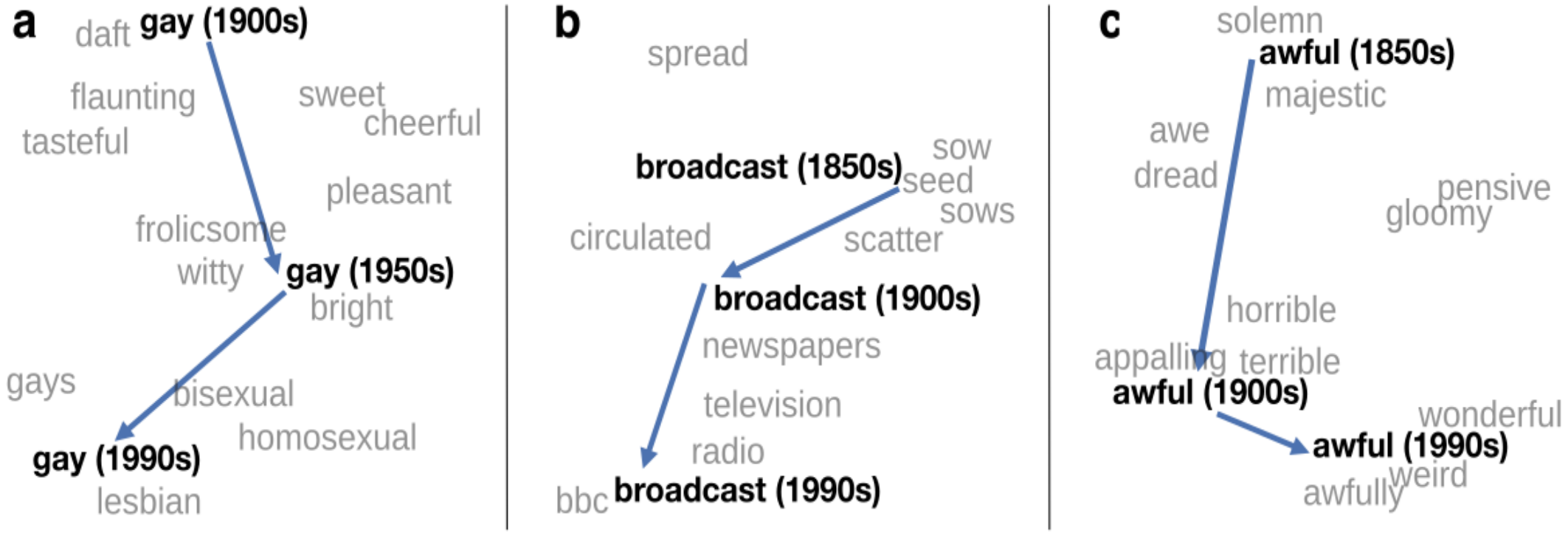
- **Skip-gram**
  - Center word -> Context words
  - Training uses Negative sampling
  - works well with a small amount of the training data
  - represents rare words or phrases quite well

- **CBOW**
  - Context words -> Center word
  - Training uses Softmax
  - Typically faster to train than the skip-gram
  - slightly better accuracy for frequent words.

Embeddings as a window onto historical semantics

- **Train embeddings on different decades of historical text to see meanings shift**

~30 million books, 1850-1990, Google Books data



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Bias in Word Embeddings

- **Embeddings can learn and amplify social biases**
  - Embeddings learn word relationships from the statistics of the text they are trained on.
  - As a result, they can also encode implicit biases and stereotypes present in that text.
- Example**: gender stereotypes**
  - Word2Vec embeddings trained on news text have been shown to produce analogies such as:
    - King – man + woman ≈ queen
  - But they can also encode undesirable associations:
    - computer programmer – man + woman ≈ homemaker
    - Father : doctor :: mother : nurse
  - Algorithms that use embeddings as part of, say, hiring searches for programmers, might lead to bias in hiring
- Gendered associations can become more pronounced in embedding space than in the underlying text statistics.

*[**Source:** Bolukbasi et al. (2016); Zhao et al. (2017); Ethayarajh et al. (2019); Jia et al. (2020)]*

Bias in Word Embeddings

- Historical embeddings can be used to **measure how cultural biases change over time**.
  - Compute a gender or ethnic bias for an adjective by comparing its similarity to:
    - words/names associated with **women vs. men**, or
    - names associated with different **ethnicities**.

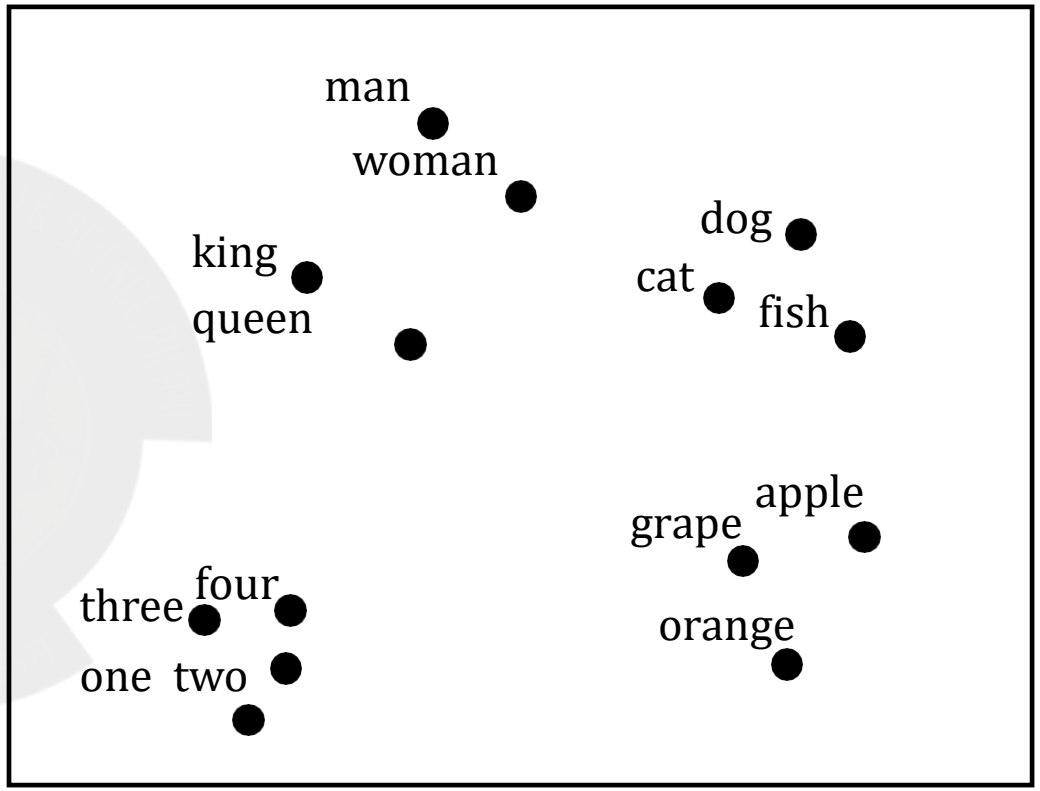
  - Competence-related adjectives ✓
    smart, wise, brilliant, resourceful, thoughtful, logical : showed a **bias toward men**, although this bias slowly decreased from the 1960s to the 1990s.

  - Dehumanizing adjectives ✓
    barbaric, monstrous, bizarre : showed a **bias toward Asians** in the 1930s, with the bias decreasing over the 20th century.
- Emerging techniques can reduce unwanted biases in embedding spaces, but they generally cannot eliminate bias completely.

Visualizing word embeddings

- **Project high-dimensional word embeddings (e.g., 100 dimensions) into 2D** so they can be inspected visually. Methods such as **t-SNE** (and **PCA**) help with such dimensionality reduction.

- Words that are close in the original embedding space tend to appear close in the 2D visualization as well, revealing **clusters and relationships among words**.



[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Analogical relations via parallelogram

- The classic parallelogram model of analogical reasoning
- To solve: *"apple is to tree as grape is to _____"*

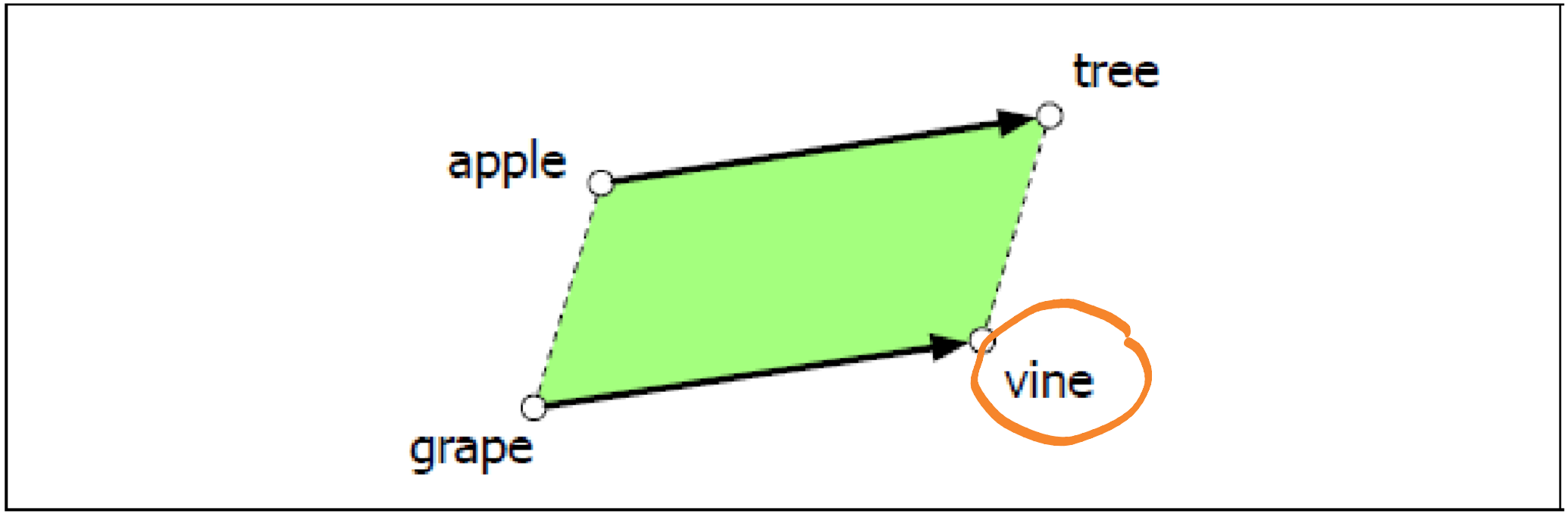


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)
  - king – man + woman is close to queen
  - Paris – France + Italy is close to Rome
- For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\text{argmax}}\ \text{distance}(x, a^* - a + b)$$

Count (frequency) based v/s Prediction based methods

Count based

- Fast to train
- Makes efficient use of observed statistics
- Primarily captures word similarity
- Large counts can dominate the representation

Prediction based

- Training Scales with corpus size
- Requires repeated training over the corpus
- Can capture richer patterns beyond similarity
- Can lead to better performance on downstream NLP tasks

GloVe: Global co-occurrence statistics with embeddings

- **GloVe = Global Vectors for Word Representation**
  - Like Word2Vec, GloVe learns dense word embeddings from a text corpus.
  - Unlike Word2Vec, GloVe starts from the global word–word co-occurrence statistics of the entire corpus.

- **The key idea:**
  - Words that occur in similar contexts should have similar representations.
  - **Frequency-based methods**
    - Corpus -> Co-occurrence counts -> Word vectors
  - **GloVe**
    - Corpus -> Global co-occurrence matrix -> Learnt dense embeddings

GloVe : Approach

- Start with global word-word co-occurrence matrix, X.
  - where X_{ij} = the number of times word j occurs in the context of word i
- let $X_i = \sum_k X_{ik}$ be the number of times *any word* appears in the context of word i
- let $P_{ij} = P(j \mid i) = X_{ij} / X_i$ be the probability that word j appears in the context of word i
- P_{ij}, thus, is the probability of seeing context word j, given target word i

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

GloVe: Approach

- Suppose we want to understand the relationship between: **ice** and **steam**
- Let's consider the ratio of probabilities P(k|ice) and P(k|steam), for different context words ***k***
- If the ratio is
  - **>> 1 ->** k is strongly associated with **ice**, but not **steam**
  - **<< 1 ->** k is strongly associated with **steam**, but not **ice**
  - **≈ 1 ->** k is similarly associated with both

Probability and Ratio	k = *solid*	k = *gas*	k = *water*	k = *fashion*
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Very small or large:
solid is related to ice but not steam, or
gas is related to steam but not ice

close to 1:
water is highly related to ice and steam, or
fashion is not related to ice or steam.

GloVe converts global co-occurrence statistics into dense word vectors by learning representations that preserve the relationships in those statistics.

References

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin
- https://jalammar.github.io/illustrated-word2vec/
- http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture02-wordvecs2.pdf
- https://arxiv.org/pdf/1301.3781.pdf
- https://nlp.stanford.edu/pubs/glove.pdf
- https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6
- www.deeplearning.ai

References

Word embedding

- https://www.youtube.com/watch?v=ERibwqs9p38
- https://www.youtube.com/watch?v=EsfNYiLVtHI&t=10s
- https://www.youtube.com/watch?v=lrPxo-92GC0
- https://www.coursera.org/lecture/nlp-sequence-models/
- https://www.youtube.com/watch?v=UqRCEmrv1gQ
- https://www.youtube.com/watch?v=g7wEfamF0Eg
- https://www.youtube.com/watch?v=Sho-b4-9ODE

Good References

- https://emerj.com/partner-content/nlp-current-applications-and-future-possibilities/
- https://venturebeat.com/2019/04/05/why-nlp-will-be-big-in-2019/
- https://www.nltk.org/book/
- https://www.coursera.org/learn/python-text-mining/home/week/1
- https://openai.com/api/
- https://analyticssteps.com/blogs/top-nlp-tools
- https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html
- https://www.cstr.ed.ac.uk/emasters/course/natural_lang.html
- https://web.stanford.edu/class/cs224u/2016/materials/cs224u-2016-intro.pdf
- https://www.mygreatlearning.com/blog/trending-natural-language-processing-applications/

Thank You!